ÉCOLE CENTRALE DES ARTS ET MANUFACTURES « ÉCOLE CENTRALE PARIS »

THÈSE

présentée par

Thomas Vosgien

pour l'obtention du

GRADE DE DOCTEUR

Spécialité : Génie Industriel

Laboratoire d'accueil : LGI – Laboratoire Génie Industriel

SUJET :

Ingénierie Systèmes basée sur les Modèles appliquée à la Gestion et l'Intégration des données de Conception et de Simulation: application aux métiers d'intégration et de simulation de systèmes aéronautiques complexes

-

Model-Based System Engineering enabling Design-Analysis Data Integration in Digital Design Environments: application to collaborative aeronautics simulation-based design process and turbojet integration studies

soutenue le : 27 janvier 2015

devant un jury composé de :

Abdelaziz BOURAS – Professeur des universités (ictQATAR) – Rapporteur
Vincent CHEUTET – Professeur des universités (INSA Lyon) – Rapporteur
Thomas NGUYEN VAN – Docteur (Snecma, Groupe SAFRAN) – Examinateur
Benoît EYNARD – Enseignant Chercheur HDR (UTC) – Co-encadrant de thèse
Marija JANKOVIC – Maitre de conférence HDR (ECP) – Co-encadrant de thèse
Jean-Claude BOCQUET – Professeur des universités (ECP) – Directeur de thèse
Chris PAREDIS – Professeur des universités (Georgia Tech) – Président du jury

2015 - ECAP0011

“We keep moving forward, opening up new doors and doing new things, because we are curious.
And curiosity keeps leading us down new paths.”

Walt Disney

REMERCIEMENTS

Je voudrais avant tout remercier Jean Claude Bocquet, directeur du Laboratoire Génie Industriel à l'Ecole Centrale Paris et directeur de cette thèse, pour son soutien, de m'avoir fait confiance pour mener à bien ces travaux et de m'avoir accueilli dans son laboratoire.

J'adresse aussi de vifs remerciements à Benoit Eynard et Marija Jankovic pour avoir bien voulu co-encadrer cette thèse. Leur aide fut précieuse pour le positionnement scientifique de ces travaux mais également pour me former à l'écriture scientifique. Je tiens aussi à les remercier pour la confiance qu'ils m'ont accordée, pour leurs encouragements et leur soutien permanent.

Je remercie également Abdelaziz Bouras et Vincent Cheutet, qui ont bien voulu être rapporteurs de ce mémoire. Merci aussi à Chris Paredis pour son regard critique sur mes travaux et qui a accepté de présider mon jury de thèse.

Un grand merci à mon encadrement Snecma et particulièrement à Thomas Nguyen Van et Arnaud Quenardel qui, grâce à leur expertise respective, ont grandement contribué à ma compréhension du contexte industriel et qui sont à l'origine de nombreux concepts présentés dans ce manuscrit. Je souhaite également remercier Pascal Graignic, autre doctorant à Snecma, pour ces trois années de collaboration et de partage pendant lesquelles nous nous sommes enrichis mutuellement. Enfin je remercie Clément Leclerc qui à travers son stage de fin d'études a développé un générateur de code qui fut très utile pendant la phase de prototypage.

Merci également aux partenaires du projet CRESCENDO avec qui j'ai eu la chance de collaborer. Je souhaite notamment remercier toute l'équipe du "Product Integration Scenario" qui ont permis de donner vie à certains concepts proposés dans ce manuscrit et en particulier Judith Crockford (AIRBUS GROUP), Vincent Tuloup et Jennifer Berquet (Dassault Systèmes), Gilles Dubourg (Siemens PLM Software), Camille Richard (Vinci Consulting) et Joao Saraipa (UNINOVA).

Merci aussi à Bernard Yannou, directeur adjoint du laboratoire LGI, pour m'avoir accueilli dans son équipe de recherche, ainsi que pour ses conseils et encouragements. Je n'oublie pas non plus le personnel (assistants, doctorants et chercheurs) du laboratoire LGI, pour m'avoir accompagné pendant ces trois ans et pour tous les bons moments passés ensemble. J'ai une pensée particulière pour Carole, Corinne, Delphine, Denise, Guillaume, Hakim, Karim, Oualid, Pascal, Sylvie, Toufik et tout ceux que j'aurai du mentionner.

Un immense merci finalement à ma famille proche; mes parents mes frères, ma sœur et ma cousine qui ont toujours été présents pour moi et qui ont fait que je suis ce que je suis aujourd'hui.

Je ne peux terminer cette page de remerciements sans remercier Inès, ma fiancée, qui a su me soutenir et m'accompagner pendant toute la durée de cette thèse. Je remercie également sa famille pour m'avoir chaleureusement accueilli et m'avoir permis de me ré-oxygéner avec l'air pur de leurs jolies montagnes.

Enfin j'adresse une pensée à mes grands parents, à ma tante et à tous les proches qui nous ont déjà quittés.

ABSTRACT

In the context of large-scale partnerships, developing complex systems (such as aeronautical products) is a collaborative and distributed work involving several domains/disciplines, teams, processes, design environments, tools and modelling languages. In such a context, engineering data have to be processed and managed in the most consistent way so as to be used by all the partners and through the different activities. System design, integration and simulation are essential phases for the verification and optimization of system capabilities. Due to the increasing complexity of aeronautical products, the Systems Engineering approach, offering multi-domain, multi-actors and multi-level system characterization, can significantly contribute to the subsystem consistency insurance within the integration phase. The main objective of the integration phase is to validate the global behaviour of a system based on carefully planned and chosen numerical simulations. Depending on the considered discipline and the kind of performed analysis, these numerical simulations require defining specific models of product architecture in order to create the required simulation models. A major issue for the integrator is to manage these models in order to identify the relevant data set to be used for the simulation and to organize this data set into a new adapted product structure and "engineering environment". Furthermore, integrating numerous components in complex system design is iterative and often produces large scale intermediate data with heterogeneous formats and multiple relationships.

During the last decade, The **Digital Mock-Up** (DMU) – supported by **Product Data Management** (PDM) systems – became a key integrated environment to exchange/share a common 3D model-based product definition between design teams. It gives to designers and downstream users (analysts) an access to the geometric definiton of product assembly. While enhancing 3D and 2D simulations in a collaborative and distributed design process, the DMU offers new opportunities for analysts to retrieve the appropriate CAD data inputs used for **Finite Element Analysis** (FEA); allowing hence to speed-up the simulation preparation process. However, current industrial DMUs suffer from several limitations among which: the lack of flexibility in terms of content and structure, the lack of digital interface objects describing the relationships between its components and a lack of integration with simulation activities and data.

The PhD introduces the concept of multi-disciplinary **digital integration chains** which are multi-level design-simulation loops where sub-systems models and data (potentially coming from several disciplines) are integrated to enable the prediction of global system behaviour, and hence verifying the compliance with expected system performances. In the context of digital integration chains, the PhD especially underlines the DMU transformations required to provide adapted DMUs that can be used as direct input for FEA of large assembly. These transformations must be consistent with the simulation context and objectives and lead to the concept of "**Product View**" applied to DMUs and to the concept of "**Behavioural Mock-Up**" (BMU). A product view defines the link between a product representation and the activity or process (performed at least by one stakeholder) that use or generate this representation. The BMU is the equivalent of the DMU for simulation data and processes. Beyond the geometric definition, which is represented in the DMU, the so-called BMU should logically link all data and models that are required to simulate the physical behaviour and properties of a single component or an assembly of components.

The key enabler for achieving the target of extending the concept of the established CAD-based DMU to the behavioural CAE-based BMU is to find a bi-directional interfacing concept between the BMU and its associated DMU. This concept is the kernel of the **Design-Analysis Integration Framework (DASIF)** proposed in this PhD. This framework might be implemented within PLM/SLM[1] environments and interoperate with both CAD-DMU and CAE-BMU environments. DASIF combines configuration data management capabilities of PDM systems with system modelling concepts of MBSE and Simulation Data Management capabilities. In PhD dissertation, the PDM and System Modelling capabilities and related concepts of DASIF are described as well as the related data model to be implemented.

This PhD has been carried out within a European research project: the CRESCENDO project which aims at delivering the **Behavioural Digital Aircraft** (BDA). The BDA concept might consist in a collaborative data exchange/sharing platform for design-simulation processes and models throughout the development life cycle of aeronautical products. Within this project, the **Product Integration Scenario** and related methodology have been defined to handle digital integration chains and to provide a test case scenario for testing DASIF concepts. Latter have been used to specify and develop a prototype of an "**Integrator Dedicated Environment**" implemented in commercial PLM/SLM applications (CATIA/SIMULIA V6 and Teamcenter for Simulation 9). These prototypes have permitted to assess the current commercial tools maturity regarding these concepts and to have a feedback regarding the feasibility of their implementation. Finally the conceptual data model of DASIF has also served as input for contributing to the definition of the Behavioural Digital Aircraft Business Object Model: the standardized data model of the BDA platform enabling **interoperability between heterogeneous PLM/SLM systems** and to which existing local design environments and new services to develop could be pluged.

[1] **PLM** for **P**roduct **L**ifecycle **M**anagement and **SLM** for **S**imulation **L**yfecycle **M**anagement

TABLE OF CONTENTS

FIGURE INDEX

TABLE INDEX

LIST OF ACRONYMS

Acronym	Complete Designation
AAM	Application Activity Model
AIM	Application Interpreted Model
AFIS	Association Francaise d'Ingénierie Système
ALM	Application Lifecycle Management
AMN	Associative Model Network
AP	Application Protocol
ARM	Application Reference Model
BC	Boundary Conditions
BDA	Behavioural Digital Aircraft
BDA BOM	Behavioural Digital Aircraft Business Object Model
BMU	Behavioural Mock-Up
BOM	Bill of Materials
BPMN	Business Process Model and Notation
BRep	Boundary Representation
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CAX	Computer Aided for X
CC	Conformance Class
CFD	Computational Fluid Dynamics
CM	Configuration Management
CPD	Collaborative Product Development
CPM	Core Product Model
CRESCENDO	Collaborative & Robust Engineering using Simulation Capability Enabling Next Design Optimization
CRUD	Create, Remove, Update, Delete (functions)
DAG	Directed Acyclic Graph
DASIF	Design-Analysis System Integration Framework
DDMU	Downstream Digital Mock-Up
DEX	Data EXchange specifications
DMD	Material ID at Snecma
DMU	Digital Mock-Up
DOF	Degree of Freedom
DSM	Design Structure Matrix
EC	Engineering Change
ECO	Engineering Change Order
EDM	Engineering Data Management
FBS	Function – Behaviour – Structure
FE	Finite Element
FEA	Finite Element Analysis
GUI	Graphical User Interface
ICD	Interface Control Document or Drawing
(I)FEM	(Integrated) Finite Element Model
IGES	Initial Graphics Exchange Specification
INCOSE	International Council on Systems Engineering
IPPS	Integrated Power Plant System
IPR	Intellectual Property Right
ISO	International Standardisation Organisation
IVVQ	Integration, Verification, Validation and Qualification

LPD	Lean product Development
MBE	Model-Based Engineering
MBD	Model-Based Design or Model-Based Definition
MBSE	Model-Based System Engineering
MDA	Model-Driven Approach
NAUO	Next Assembly Usage Occurrence
NIST	National Institute of Standards and Technology
OAM	Open Assembly Model
OCC	Open Cascade CAD (library)
OMG	Object Management Group
PDM	Product Data Management
PDP	Product Development Process
PLCS	Product Life Cycle Support
PLM	Product Lifecycle Management
PPI	Power Plant Integration
PPS	Power Plant System
RDF	Resource Description Framework
RDMU	Referential Digital Mock-Up
RFLP	Requirements, Functional, Logical and Physical
SADT	Structured Analysis and Design Technique
SBD	Simulation-Based Design
SDM	Simulation Data Management
SLM	Simulation Lifecycle Management
SE	Systems Engineering
SoS	Systems of Systems
STEP	STandard for the Exchange of Product model data
UML	Unified Modelling Language
UoF	Unit of Functionality
V&V	Verification and Validation
VSM	Value Stream Mapping
XMI	XML Metadata Interchange
XML	eXtensible Mark-up Language

Chapter 1: General Introduction

1.1 Introduction and research context

The research work presented in this dissertation has been carried out within the Integration Division of the Snecma company which belongs to the SAFRAN Group. This work is done in collaboration with the Industrial Engineering Laboratory (LGI) of the Ecole Centrale de Paris (ECP).

Conssidering nowadays context of strong competitiveness, European aircraft, engine and equipment manufacturers are facing greater challenges than ever before. The market demands that more complex products are developed with shorter lead times and more cost effectiveness. Therefore, the reduction of the time to market has become a strategic variable for firms, particularly for manufacturers of complex systems such as aeronautical products. Nowadays, aeronautics and aerospace programs have evolved towards large-scale partnerships. The development of these complex products/systems is hence a collaborative and distributed work involving several domains/disciplines, teams, processes, design environments, tools and modelling languages. In this context, engineering data have to be processed and managed in the most consistent way so as to be used by all the partners and through the different activities.

In such a context, the PhD has also carried out during the European FP7 project called CRESCENDO (Collaborative & Robust Engineering using Simulation Capability Enabling Next Design Optimization). This project aims at developing methodological approaches and tools in order to support channels of digital integration in the aeronautical extended enterprise through a collaborative digital platform called the Behavioural Digital Aircraft (BDA). Indeed, aeronautics and more especially product development activities have been impacted in recent years by the advent of the use of **digital engineering technologies.** This PhD was hence initiated to investigate the use of digital technologies within collaborative product development processes in the aeronautical extended enterprise and specifically within integration phases.

The aim of this PhD is to contribute to the improvement of design, integration and simulation activities in aeronautics, but more generally in the context of collaborative complex product development. This objective is expected to be achieved through the use and improvement of digital engineering capabilities. These capabilities need to fulfil the needs of the various engineering business processes and actors using them. Moreover, the needs to ensure the continuity of information between working teams, the data exchange efficiency, the interoperability between systems and the control through an integrated reference framework for collaborative product development have rapidly appeared while analyzing the industrial context.

This PhD introduces our investigations for developing **an integrated reference framework** to ensure a better integration between design and analysis data. This integration will support the needed “analysis product views” regarding the scope and objectives of the various performed analyses. This framework and related new digital engineering capabilities (engineering data management and computer-aided technologies) also aim at supporting the definition of product architectures so as to organize and facilitate modelling and simulation activities. This framework also supports the specification of system interfaces, hence enabling a better integration of sub-system models and of several product behaviour simulations. The assessment and the use of standards for data exchange in this study also complies with very constraining interoperability issues encountered by engineers while using the digital engineering technologies which support their design and simulation activities.

1.2 Research methodology and process

Our research methodology and process was inspired by the one proposed in []. Our research work has been guided and structured by four major stages (cf. Figure 1):

- Part I: the problem statement phase which mainly consisted in performing an audit and an analysis of the industrial context as observed at Snecma and as discussed with Crescendo industrial partners. This phase permitted first to better clarify the PhD scope and objectives and secondly to identify the industrial problems and requirements.
- Part II: the state-of-the-art phase which has consisted first to derive our industrial requirements into high-level research questions to define our scientific positioning and the state-of-the-art focus. This phase ends by a gap analysis of the existing research works regarding industrial requirements and by the identification of our potential PhD contribution.
- Part III: the concepts proposal and development phase in which we introduce all concepts and related capabilities proposed in order to meet the industrial requirements and to further develop and implement concepts already previously defined.
- Part IV: the demonstration and industrial validation phase which lead to an assessment study and gap analysis regarding the implementation of the proposed concepts (proof of concepts). Finally the PhD ends by the identification of new open perspectives for future research work and development.

The circle which appears in Figure 1 represents our scientific positioning and contributions regarding three main research areas that have been identified for bridging the gap between design and simulation for efficient system integration (see section 5.2).

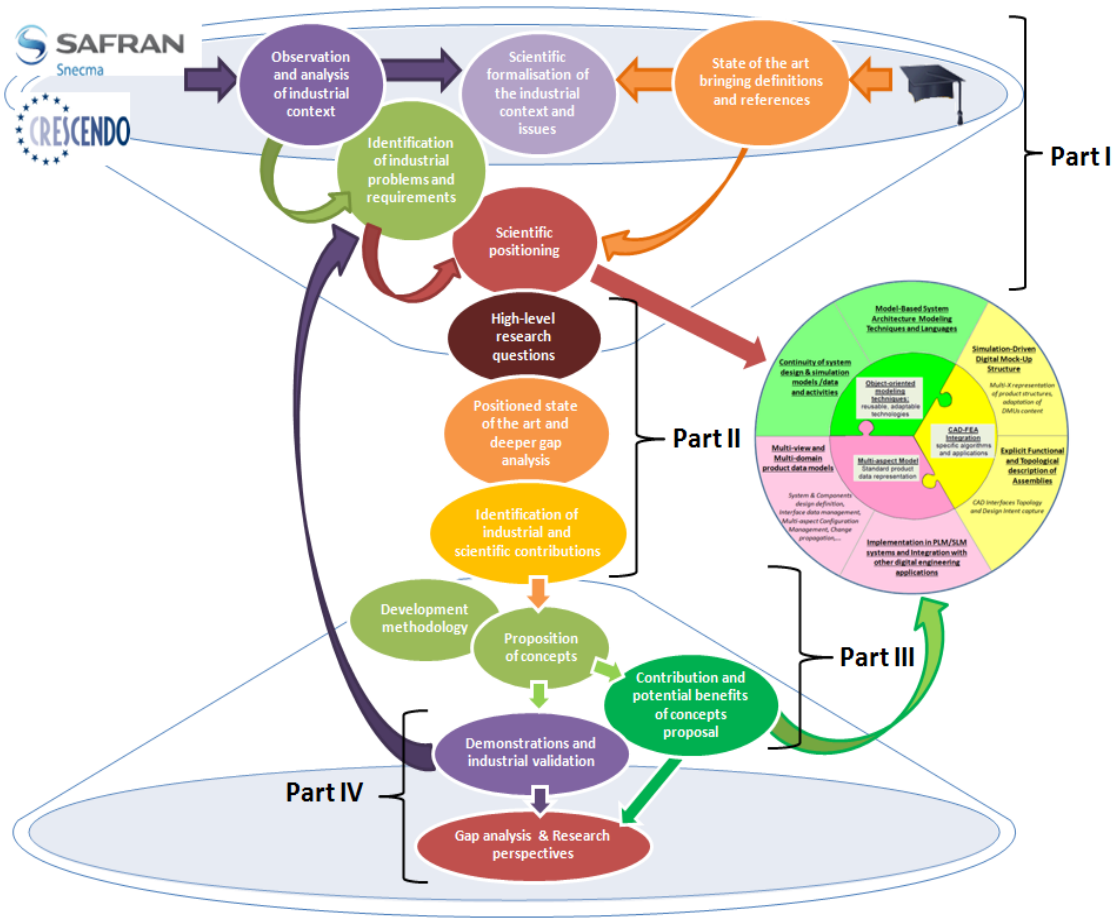


Figure 1: Research methodology and structure of the PhD thesis

PART I: PROBLEM STATEMENT

Currently two main limitations that currently impact the efficiency of engineering design activities in the aeronautic industry have been identified in this research work: 1) the complexity of the product itself but also its impact on the complexity of the related organisation and process to develop it; 2) the lack of a "central reference" for design, integration and simulation activities to make the right data and information available at the right time and for the right actor to perform efficiently these activities.

Concerning the first limitation, the aero-engine systems, operating in a very constrained environment, are not only complex because of the number of their components and interactions, but also because these components and interactions have to satisfy many multi-disciplinary functional requirements at the same time. The validation of these functional requirements requires the involvement and coordination of several inter-dependant simulation disciplines (mechanical, aerodynamics…) at different system breakdown levels. As a result aeronautical companies face increasing needs in simulating not only standalone components but also large assemblies containing up to thousands of components (such as aero-engines). In the context of collaborative and distributed design, integrating and validating these subsets is great challenge because it requires synchronizing inter-dependant processes and related data, but it also requires providing appropriate "product definition views" (characterising the ideal content and organization of product data for a discipline).

The multi-disciplinary nature of complex system projects like aeronautical programs results in large quantities of design data, managed in different tools used in various application domains. Moreover, aeronautics projects have evolved through large-scale partnership and with the advent use of computer-aided applications, a large amount of data is then produced by the different partners, co-designers through the various involved engineering disciplines. In this context, product data has to be processed and managed in the most consistent way so as to be used by the different partners and through the different activities [Nguyen Van et al., 2006a]. Since the 90's, Product Data Management (PDM) tools have appeared and provided a strong support to address this challenge. However, and as it will be further explained in this section, the mentioned "central reference" or "common referential framework" is still missing in these PDM systems, especially while addressing the issue of integrating design and simulation/analysis data.

This entire part of this PhD thesis is based on the observation and analysis of the industrial context. This PhD has been undertaken within the SNECMA company (SAFRAN group) and more precisely within the Power Plant System Integration Division. It has also been carried out within a European research project: the CRESCENDO project. CRESCENDO means **C**ollaborative & **R**obust **E**ngineering using **S**imulation **C**apability **E**nabling **N**ext **D**esign **O**ptimization. This European consortium involves 59 partners representing a cross section of European aeronautics. The project aims at delivering the modelling and simulation backbone of the aeronautical extended enterprise: the **Behavioural Digital Aircraft** (BDA). The BDA concept might consist in a collaborative data exchange/sharing platform for design-simulation processes and models throughout the development life cycle of aeronautics products.

The first chapter of this part introduces:

- the contextual changes and industrial stakes of the aeronautics industry and the key strategic variables of aeronautical product development programs,
- the characterisation of an aircraft's power plant system complexity and the business and organisational impacts of this complexity on engineering design activities,
- the related system engineering and integration challenges to manage efficiently this complexity and the importance of handling efficiently design-simulation loops.

Therefore, the second chapter of this part is dedicated to the analysis of the collaborative digital engineering environments and related challenges to support efficiently design-simulation loops. This analysis is exposed in Chapter 3 and includes:

- An overview of the various digital design environments and related industrial practices,
- An introduction to digital integration chains and an emphasis on the role and potential usages of the Digital Mock-Up (DMU) in collaborative simulation-based design.

Finally, based on all these observations, chapter 4 underlines the limits encountered while using data extracted from current DMU and Engineering Data Management environments and synthesises the key identified industrial requirements.

Chapter 2: Industrial context and challenges

Before introducing the specific industrial challenges addressed by this PhD, it is important to remind briefly the key strategic variables of aeronautics product development programs; it is the aim of the first section of this chapter. The second section aims at characterising the complexity of an aircraft's power plant system complexity and the business and organisational impacts of this complexity on engineering design activities. Third section introduces the notions of System Engineering and System Integration and underlines the challenges to manage efficiently this complexity as well as the importance of handling efficiently design-simulation loops. Finally a conclusion summarizes the industrial challenge addressed by this PhD.

2.1 Particularities of the aeronautics industry: contextual changes and industrial stakes

Until the end of the 1980's, the aeronautics and aerospace industry was characterized by a dominant emphasis on the performance of systems rather than on time or cost to develop and sustain the systems. From the 1960's until the 1990's, the time required to develop aeronautics and aerospace systems, increased (by 80% for American DoD systems [McNutt, 1999]). Several authors have stated or even demonstrated that the root causes for these time increases are growing project, process and product complexity [Clift&Vandenbosch, 1999] [Murman et al., 2000] [Kim&Wilemon, 2003].

By the 1990s, with an industry facing global competition in both commercial and military markets, all the aeronautics and aerospace sectors try to develop their systems "*Better, Faster, Cheaper*" [Murman et al., 2000]. In 1991, Clark and Fujimoto define the three main outcomes and performance dimensions of the product development process that affect the ability of a product to attract and satisfy customers [Clark&Fujimoto, 1991]:

- the total product quality: the extent to which a product satisfy customer requirements,
- the **lead-time** or **time-to-market**: the measure of how quickly a company can move from concept to market. By reducing time to market, companies can deliver a product to market before their competitors, thereby capturing market share and expand the number of new products they develop.
- the productivity: the level of resources required to take the project from concept to market; this dimension directly impacts the product development cost and time.

In the last decade (from 2000 to 2010) the trend has been inversed since aeronautics and aerospace companies have made great efforts to reduce their development times in order to gain competitive advantages and to quickly meet their customers changing needs with high quality and low cost products. Indeed, for companies developing complex systems, time-to-market is largely impacted by the development cycle time [Griffin, 1997]. The focus on the reduction on development time is seen as an organizing focus from which to organize development efforts. They have found that by focusing on the reduction of development time, they force improvements in their business processes [Clark&Fujimoto, 1991].

Therefore PDP and its cycle time largely impact time-to-market. This statement is particularly true for complex system/product to design (see section 2.2.1). Several authors have demonstrated the negative impact of project and product complexity on the development cycle time and cost [Griffin, 1993]

[Griffin, 1997] [Meyer&Utterback, 1995] [Kim&Wilemon, 2003]. It is hence essential, before addressing the particularities of aeronautics programs and identifying the potential opportunities to improve PDP efficiency, to better apprehend the complexity of aeronautics products (particularly aero-engines) and what are the business and organisational impacts of this complexity on the way of managing PDP activities.

2.2 Design and integration of complex aeronautics system

2.2.1 Definition of the Complex System

According to Weinberg and other system theorists, a system describes a specific way to look at the world considering its components as part of a whole [Weinberg, 1975]. In this whole, each component's features and behaviours result from the features of the organized interactions that unify these components to make the system exist [Capra, 1997]. **One main characteristic of this vision is the ability to move our attention between systems levels and apply similar concepts to different systemic levels.** For this PhD thesis and from an engineering perspective, the following definition of a system is used, based upon the definitions of [Pahl et al., 2007], [Lindemann et al., 2009] and [NASA, 2007]:

A **system** is a construct or collection of different technical artefacts that are artificial, concrete, mostly dynamic, and consist of ordered elements, which interrelated, produce results not obtainable by the artefacts alone. The results include system-level qualities, properties, characteristics, functions, behaviour, and performance. The value added by the system as a whole and its behaviour, to which the parts contribute independently, is primarily created by the relationship among the parts; that is to say, how they are interconnected. Systems are delimited by boundaries and connected to their surroundings or other external systems by inputs and outputs. Changes to parts of a system or modifications of their features and/or parameters during a time period characterize what is called the "system dynamic" distorting the whole system behaviour and its stability during this time period.

This dynamic is one of the major factors of system complexity. The adjective "complex" is a notion that often get mixed up with the adjective "complicated". A complicated system is a large system (many components) that encompasses many parameters with intense connectivity [Lindemann et al., 2009]. **A system**, complicated or not, **becomes complex when its system or parts' parameters and interactions are subjected to high dynamic of change.** In [Suh, 2005a] [Suh, 2005b], Suh defines the complexity as the measure of uncertainty in achieving the functional requirements of a system within their specified design range. **When there are many functional requirements that a system must satisfy at the same time, the complexity of the system is determined by whether or not the design parameters chosen to satisfy the functional requirements couple functional requirements to each other.**

Based on these definitions, next section aims at characterizing and analyzing the complexity of an aircraft power plant system. This complexity is analyzed from two perspectives:

- The **static complexity**: characterized by number of interactions between system's constituents and static complexity of these interactions;
- The **dynamic complexity**: characterized by the dynamics of interactions between system's constituents in the various operational states of the system and the number of multi-disciplinary functional requirements to fulfil and related behaviours to study in the same time.

2.2.2 Definition of complexity for an aircraft Power Plant System

Before the development of the turboprop technology, turbojet engines only supplied the propulsion to the aircraft, with speed and reliability as main customer requirements. Nowadays, reliability is still a key requirement but the speed criterion has been replaced by the efficiency ratio. An aero-engine with a modern turbofan, provides not only the motorisation of the aircraft but also supply the aircraft manufacturer with three vital elements for the aircraft: the electric energy, the air required for pressurizing the cabin and for starting the engines, as well as the hydraulic power needed to operate the various equipments of the aircraft. Integrated with the aircraft, an aero-engine is part of a higher system called Integrated Power Plant System (IPPS). In the case of an assembly under the wing, an IPPS consists of the main elements that are the turbofan (1), the nacelle (2), the suspensions (3), the various equipments (4) and the interface with the aircraft: the pylon (6 and 7) (see Figure 2 below).

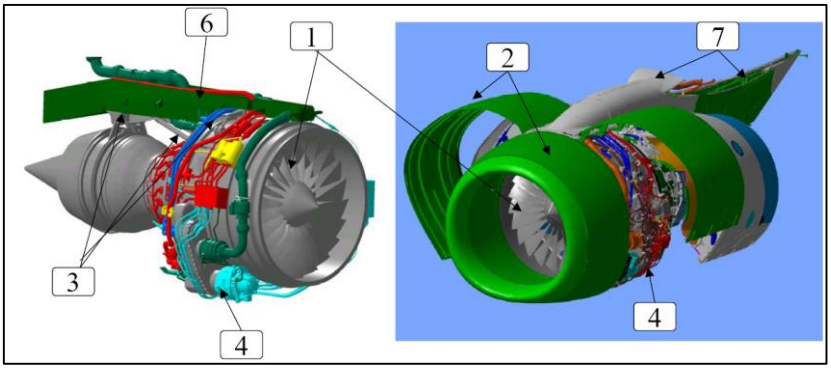


Figure 2 : Simplified 3D digital mock-up of an IPPS

The development of an aero-engine requires deep technical knowledge in very specialized scientific disciplines among which aerodynamics, structural mechanics, aero-acoustics, fluid mechanics, thermodynamics, materials science, etc. The turbo machines should endure intense thermal, mechanical and vibratory stresses and meet high operating constraints.

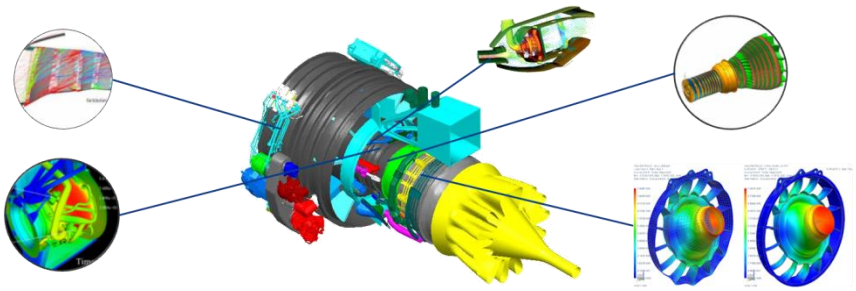


Figure 3: Multi-disciplinarily nature of an aero-engine

The dynamic complexity of an aero-engine is characterized by the dynamics of interactions between system's constituents and combination of possible paths through the various operational states of the system. It highlights the impossibility of a complete validation/verification of all the possible scenarios by combining all possible behaviours of the modules and their interactions. The **effects of couplings** (multiple simultaneous physical phenomena) that may emerge in operation are also important complexity drivers. They can change the properties of a hardware module due to thermal effect related to the proximity of a heating flow (thermo-mechanical coupling) or electromagnetic effects. The prediction of these potential coupling behaviours is also of primary importance for efficient system integration.

Today, the **static complexity** of an IPPS is characterized by the large number of subsystems and components (more than 10 000 thousands of parts) interacting together via a large number of interfaces that need to be defined, specified and validated all along the development life cycle.

Interfaces define the physical relationships between product components. They represent the physical or theoretical boundaries where two or more system components meet and interact and where domain-specific applied rules and conventions define their interaction and related design intent.

Interactions are the physical phenomena that occur at the interfaces between connected components. Their definition specifies the domain-specific functional, physical, behavioural and semantic features that define the rules and conventions to apply. These rules and conventions concern domain-specific physical features (mechanical, electrical, thermal, etc.) semantic or functional features as well as potential information exchange.

The scheme by which the sub-systems, components and interfaces are arranged is called **"**product architecture" or "product structure".

Product architecture is the scheme by which the functional elements of the product are arranged into physical chunks (building blocks or modules) and by which the chunks interact [Ulrich et al., 2011].

The definition and choice of product architectures are fundamentals in preliminary design stages since it provides the basis to organize PDP activities, define partnerships actors and rules as well as to manage/reduce system complexity [Yassine et al., 2003] [Sosa et al., 2004] through the concept of modularity.

Modularity is defined as the degree to which a product's architecture is composed of modules with minimal interactions between modules [Gershenson et al., 2003]. Ulrich and Eppinger even state that modularity is the most important characteristic of product architecture.

The most modular architecture is one in which each functional element of the product is implemented by exactly one chunk (subassembly) and in which there are few interactions between chunks [Ulrich et al., 2011]. Modularisation introduces challenges for integrating the system modules by managing efficiently interface/interactions and identifying impacts between sub-systems on the behavioural level. For instance and as shown on Figure 4, in large-scale projects like an aero-engine development program, products are decomposed into functional modules, sub-modules and so on, and work-packages of the studies on this product are externalized to partners and/or sub-contractor [Eynard&Yan, 2008]. Design teams are no longer working alone on the design of a module but perform it within the whole product environment and in interaction with the work of others [Fuh&Li, 2005].

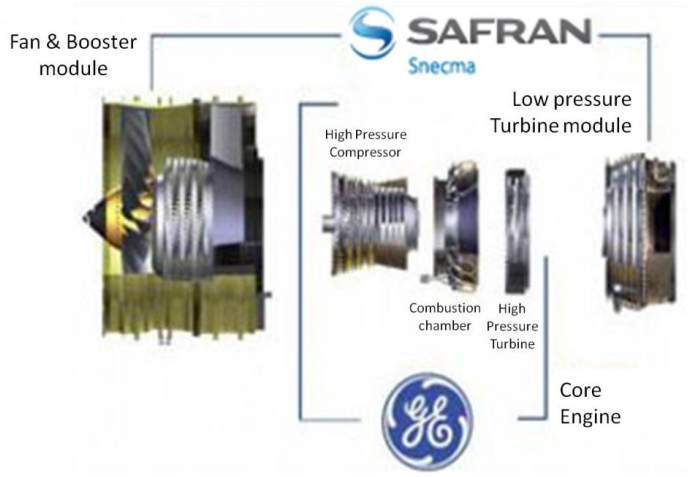


Figure 4 : GE-Snecma modular work sharing for CFM56 engines development

New aero-engine concepts and architectures are being explored to reduce mass, fuel consumption, development cost and environmental impact while increasing performance [Sandberg et al., 2009]. As a result, the main ongoing requirements of future IPPS are defined by a reduction in CO2 emissions and noise. The innovative concepts and architectures proposed for the medium term are positioned according to these two variables on Figure 5.

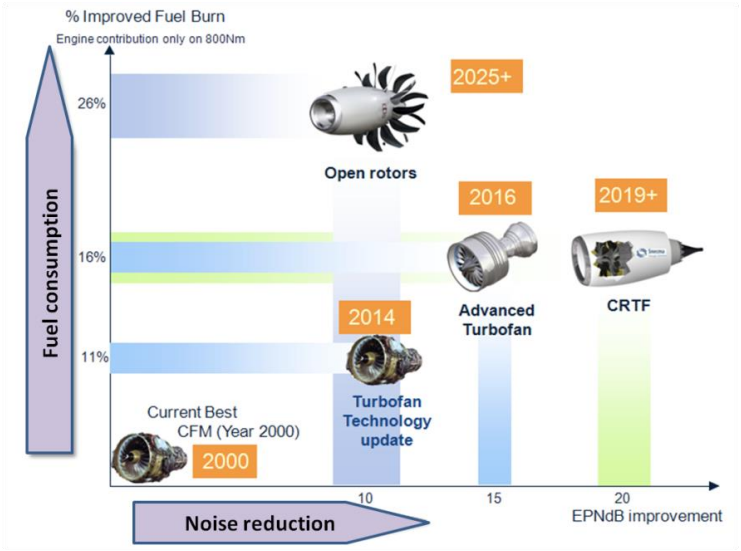


Figure 5: New IPPS concepts positioned regarding their environmental impacts

This significant change can only be done in very close and early cooperation with the design of the aircraft. Indeed, new engine architectures will have impacts on the way of interfacing with the aircraft and will probably lead to new aircraft architectures. However this kind of projects are subjected to high risk and uncertainties since they require a lot of development resources investment without having any guaranty to get a better product or to not increase the product complexity by not anticipating new unknown design constraints. The high competition in the aeronautics sector and the time-to-market pressure are real barriers to such innovative projects.

That is why **world-class aeronautic leaders need to develop very strong partnerships and relationships in order to reduce technological, financial, and market risks, as well as to pooling best-in-class competencies and sharing development and operational costs**. [Esposito, 2004] and [Pritchard&MacPherson, 2007] highlight the peculiarity of the aircraft sector characterized by “the ex-

istence of a complex network of long-term relationships having an evolutionary nature where collaboration and competition exist hand in hand and where the stability of the network is guaranteed by its constant change". In order to survive and gain competitive advantages in such a business environment, companies are increasingly motivated and/or forced into more extensive interactions with its surroundings (e.g. suppliers and customers); they are establishing/joining extended enterprises.

2.2.3 Collaborative PDP in the aeronautics extended enterprise

The extended enterprise notion is based on a product-oriented strategy aiming at increasing the effectiveness and decreasing the time of the product development while integrating both product and process requirements as early as possible. The aim of this organisation is to use available competencies, both within and outside the company [Benchimol, 1993]:

- Inside: through the different entities which belong to the company,
- Outside: through the different partners' firms, suppliers, sub-contractors, distributers, customers etc. composing what is called "the extended enterprise".

We can thus distinguish three main types of entities that can be involved in the extended enterprise: the source mother enterprise, the internal sub-contractors, the external sub-contractors and project partners. The product is decomposed into several modules which developments and studies are distributed to partners and sub-contractors. According to both functional and physical breakdowns of the system and its sub-systems or modules, multiple layers of collaboration appear [Nguyen Van et al., 2006a] (see Figure 6 below).

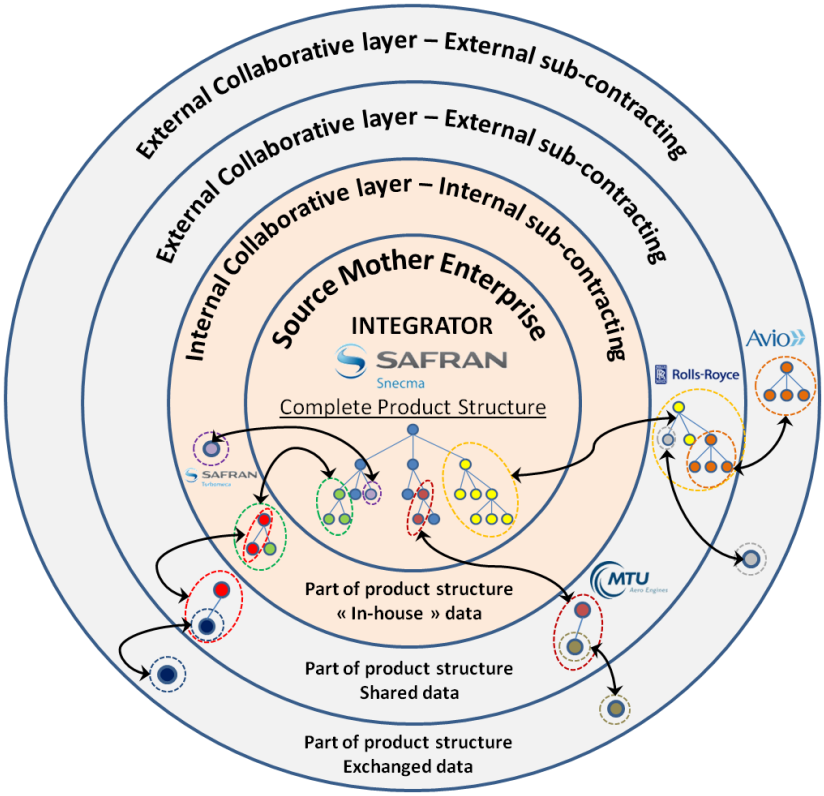


Figure 6: Multi-layer and multi-partner work breakdown into design packages. Adapted from [Nguyen Van, 2006]

In this configuration, multi-partner development programs rely on the integration of design work and data between the different partners. Therefore the extended enterprise requires a deep change within the enterprise organization and needs adaptation of bilateral agreements with its partners (Cus-

tomer or Supplier) to implement partnerships in specific areas. This new collaborative way for developing complex systems induces changes in designing, producing, operating, maintaining and disposing of systems.

Collaborative Product Development (CPD) is seen as the application of team-collaboration practices to total product development efforts. According to Willaert *et al*, CPD is a systematic approach to control life cycle cost, product quality and time to market during product development by concurrently developing products and their related processes with response to customer expectations [Willaert et al., 1998]. The difficulties that are underpinned in the collaborative tasks of PDP come as well from the application of the principles of systems engineering, management of cross-functional departments and of course linked strategy of developing the program portfolio management [Davis et al., 2004].

Although PDP models and design practices differ between companies and professional habits, **PDP is always iterative and sequenced in phases** [Pahl et al., 2007] implying the necessity to introduce validation steps/gates and segment PDP. According to what is done currently at Snecma, the engine development life cycle is also divided in four main phases (see Figure 7 below). Figure 7 also displays the milestones that structure the engine life cycle. To progress in its lifecycle, the engine must be promoted at each milestone (e.g. RFP: Request for Proposal; FETT: First Engine To Test, Certification, etc.).

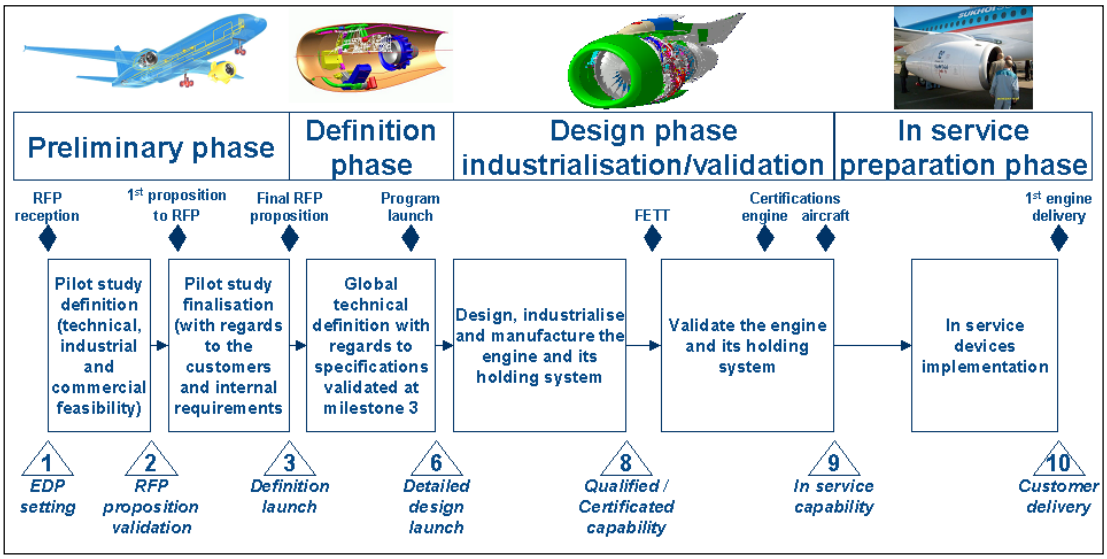


Figure 7 : Engine development life cycle [Nguyen Van, 2006]

Each phase corresponds to a specific state of the engine:

- The **preliminary phase**: this stage is a pre-definition study of the product. The first product concepts are established and the commitments on cost & project plan are defined. The phase ends by the choice of a robust architecture, compliant with the airframe manufacturer's requirements but also driven by internal requirements. This phase corresponds to the aggregation of the clarification task and Conceptual Design phase introduced above.
- The **definition phase**: this phase validates the previous one. Engine specifications are detailed and the product architecture is validated for further studies and design. This phase ends by the freeze of the IPPS architecture and the technical specifications. The sub-systems and main equipments are then available with the appropriate level of design to guarantee an optimal

performance stack-up and sufficient interface definitions. This phase corresponds to the embodiment phase.

- The **detailed design phase** (include industrialization and validation): this is the phase where the engine is being designed, virtually simulated and validated, manufactured and finally tested. All activities are synchronized and many loops can be performed to define the product in details. At the end of this phase all requirements must be demonstrated, the design validated and the airworthiness requirements demonstrated and approved.
- The **in service preparation phase**: the last phase before delivery. The engine is completely manufactured and prepared in order to be delivered to customers.

Nowadays, CPD is characterized by "**Simultaneous Engineering**" (performing different process steps at the same time) and "**Concurrent Engineering**" (developing neighbouring components in parallel). The aim of concurrent and simultaneous engineering is to parallelise activities in order to reduce the product development time and cost by enabling overlapping of design activities and phases of the development life cycle. However the difficulty is to coordinate design activities and information exchange between partners and co-designers. Communication, negotiation, coordination and cooperation are essential in order to use systematic cross functional processes, methods and tools within the extended enterprise [Pardessus, 2001]. In the case of CPD and simultaneous engineering, the first one is characterized by an overlapping of design processes and activities. For instance, an aero-engine is part of a higher system (the IPPS) which is itself part of an even higher system: the aircraft. As a consequence, the first customer of the engine manufacturer is the aircraft manufacturer. Both have a specific life cycle to follow the development of their product. Figure 8 shows both life cycles in parallel. Each phase has specific milestones to overcome. Similar to the engine life cycle phase those phases have specific meaning for the aircraft life cycle.

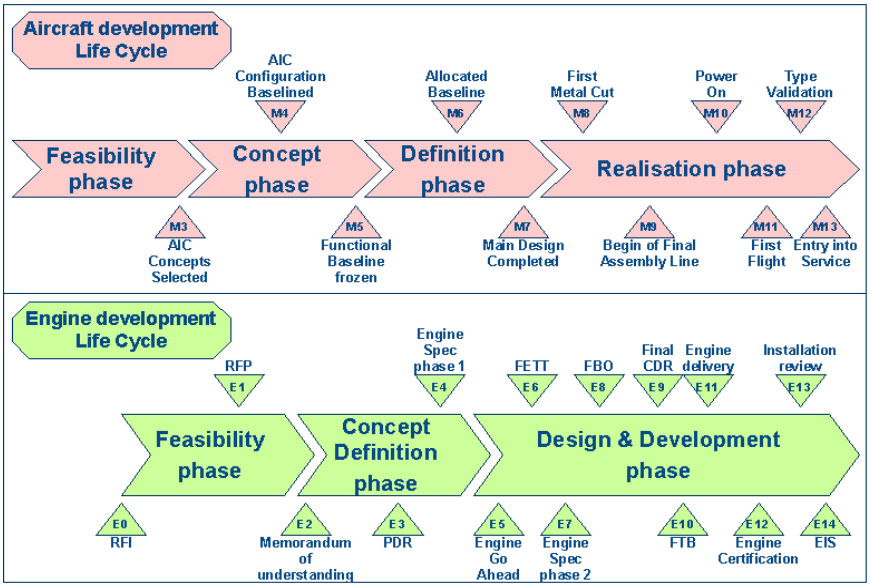


Figure 8: Engine and Aircraft development cycle in parallel [Nguyen Van, 2006]

For a product as complex as an engine, many levels of life cycles exist. Indeed, each partner is in charge of a product that is part of a higher product. All those life cycles must be synchronized to ensure an efficient integration of the sub-systems. The harmonization and synchronization between development of the engine and the aircraft are hence a critical issue to ensure a delivery on time of the final product to the final customer. To overcome this issue the different partners often define an integration

planning between stages. The integration planning is considered to be the basis of the collaboration. All along the aircraft development, communication with the engine development is therefore very important to ensure a good integration and avoid delays, design mistakes and rework. This integration involves the exchange of engineering data and information between partners all along the development life cycle. The large amount of data created by the different partners has to be processed and managed in the most consistent way so as to be used by the different partners, at the right time, and through the different activities. Hence, there is an increasing need of creating referential and associated processes for data management purpose from the early phases of a project.

As System Integration is crucial to successful CPD and simultaneous engineering, it is necessary to have a finer description of the different level of design granularity and the various disciplines and interrelated processes to which the systems and sub-systems are designed and integrated during this development life cycle. System Engineering processes and methods can help to efficiently manage these multi-domain, multi-actors and multi-level system characterization. Next section gives an overview of System Engineering and System Integration notions in order to further detail the analysis of the related issues and challenges.

2.3 Challenges of System Engineering and Integration

2.3.1 Definitions

Systems Engineering (SE) is an interdisciplinary and methodological approach that encompasses all activities appropriate to design, develop, make evolve and test a set of products, processes and people skills, providing an economical and efficient solution to the needs of stakeholders and that is acceptable by all [IEEE, 2005] [NASA, 2007]. This set is integrated in a system, by continuously searching for balance and optimization throughout its life cycle [AFIS, 2009].

A variety of SE process standards have been proposed by different international standards. The purpose of each major SE process model standard can be summarized as follows:

- ISO/IEC 15288 – Establish a common framework for describing the lifecycle of systems [ISO, 2008].
- ANSI/EIA 632 – Provide an integrated set of fundamental processes to aid a developer in the engineering or re-engineering of a system [ANSI, 1999].
- IEEE 1220 – Provide a standard for managing a system [IEEE, 2005].

System Integration consists in building a system by a progressively assembling the systems' components following an incremental and systematic process properly planned. This process results in integrating the systems' components that have already been validated, checking each time the interfacing conditions and the behaviour compliance (functions and performance) of the obtained assembly [Meinadier, 1998]. This is done until obtaining the complete integrated and validated system. The integration ascending branch is generally based on a V&V (Verification and Validation) or IVVQ (Integration, Verification, Validation and Qualification) approach [Haskins&Forsberg, 2011] [AFIS, 2009].

2.3.2 System integration : a simulation-based design process

System integration aims at validating the global system behaviour through carefully planned and chosen interdependent numerical simulations (see Figure 9). Numerical simulation refers to the use of computational models to analyze and evaluate the behaviour of a designed system. Both in design and integration phases, numerical simulations gradually became one of the key industrial resource to reduce the design cycles while ensuring the improvement of the quality and the performances of the products [Charles et al., 2005]. The numerical/digital simulation is not a stand-alone process. It is an iterative approach where loop of simulation aims to validate and to optimize the physical behaviour of a principle solution. We have observed that simulations are very important performance and hence value or waste drivers in an aero-engine development life cycle for the following reasons:

- Performed digital simulations provide an approximate (depends on the detail degree of models) knowledge on the product behaviour;
- Digital simulations of complex models require many data inputs which are difficult to obtain and the overall simulation process is very time consuming, and hence may be delayed because of inaccessible data (if inputs come from other simulations results that do not exist or that are not accessible), unsuitable or outdated data (due to simulation tools interoperability capabilities, or from lack of coordination rules between design offices);
- The more accurate virtual prototyping and simulation are performed, the less testing prototypes are needed. Aeronautics and aerospace products are subjected to very strict certification rules and are obliged to perform a lot of expensive certification tests. The physical prototypes used for these certification tests are expensive, and hence increase development and product costs.

For these reasons, a major part of PDP in high value-added industries has become a **Simulation-Based Design** (SBD) process where simulation is the primary means of design evaluation and verification. When coupled with appropriate validation processes executed during the development of a simulation-based design system, the resulting capabilities can provide companies with the ability to design superior products in less time and at lower costs [Shephard et al., 2004].

As observed and shown in Figure 9, all along an IPPS development life cycle, numerical simulations are performed in every scientific domain and at each system breakdown level in order to enable detailed exploration of the design space and significant performance improvements. In addition to these disciplinary-centred requirements, the tight coupling of several phenomena interacting together makes it a huge challenge to find a global optimum for the entire system, which can be very far from a collection of single-discipline optimizations. The search for this optimal performance stresses the need to adopt an **integrated design approach**: it is desirable to design components using a system (whole aero-engine or whole aircraft system) approach in order to optimize component design for system-level performance [Defoort et al., 2012].Two simulations are inter-dependant when the results of one of them are used as input by the other one. There are two kinds of simulation inter-dependencies:

- Multi-domain inter-dependency between simulations performed at the same system breakdown level: due to the coupling of physical phenomena interacting together (e.g. a mechanical analysis uses the temperature and pressure fields resulted from the thermal analysis);
- Multi-level inter-dependency between single-domain simulations performed at different system breakdown levels: the simulation results of the sub-system interfaces of the studied system are transferred to the external interfaces of each sub-system and used as boundary conditions for the sub-systems simulations.

The models used for these various simulations have not the same level of abstraction depending on the breakdown level and the solver algorithms used to perform the simulation. As a result the time to create the models and the final simulation lifecycle times differ from one discipline to another. Therefore, the synchronisation of these multi-domain and multi-level interdependent simulations, performed by different partners using different tools, represents a big integration challenge. These time gaps between analysis studies generate also some knowledge gaps and hence risk and uncertainties in PDP.

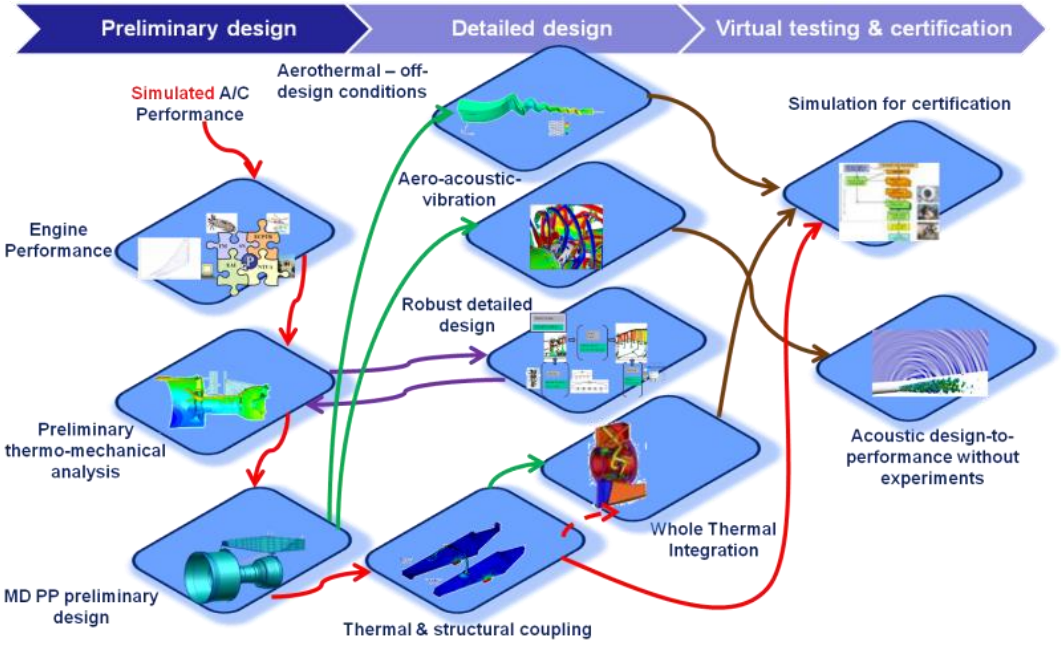


Figure 9: Examples of inter-dependant numerical simulation to perform IPPS integration along the development lifecycle

From a micro perspective, Figure 10 illustrates these knowledge and time gaps within an aero-engine development life cycle. In red is represented the product definition life cycle (the evolution of design definition (geometries, design parameters, bill of materials with mass properties, etc.). In blue is represented the mechanical analyses life cycle that correspond to the successive mechanical simulation and in green the Aerothermal analyses life cycle.

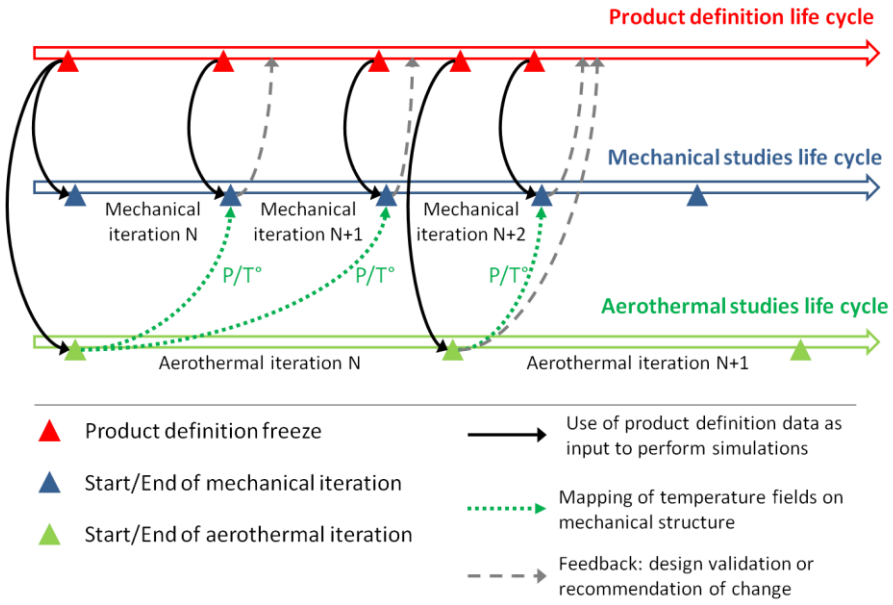


Figure 10: Illustration of time and knowledge gaps from a micro perspective: non-synchronisation of mechanical and aerothermal studies in an aero-engine development life cycle

In current practices, performing an analysis require freezing the product definition until the next iteration. An aerothermal analysis iteration (time between two analyses of the same system or sub-system) is about two times longer than a mechanical analysis. This is due to the fact that aerothermal simulation models are much longer to create. Moreover, mechanical analyses use the results of aero-thermal analyses (pressure and temperature fields) to take into account the deformation caused by the pressure of the fluid on the structure or the modifications of material properties dependant of the temperature. The consequence is that, when starting the mechanical iteration N, the mechanical engineer use the Aerothermal analysis results from the Aerothermal iteration N-X (X depending on the time gap between a mechanical and an aerothermal iteration) that have been calculated from a product definition freeze which is different from the one used by the mechanical engineer for the starting iteration. As a result, the new mechanical analyses may be based on wrong assumptions and there is a risk to be obliged to perform new avoidable iterations (rework). This is an example of the knowledge and time gaps that can emerge in complex PDP. These gaps are often unavoidable because simulation models and tools are specific to the discipline addressed. However, the gaps are also due to the time required to get the relevant product definition view and data inputs used to create the simulation model. Therefore the reduction of these gaps and uncertainties is still an ongoing challenge that can be addressed by fastening the design-analysis iterations.

2.3.3 The design-analysis integration challenge

Previous section has underlined an important issue related to complex system engineering and integration challenges: coordinating efficiently inter-dependant and multi-disciplinary simulation activities. Simulation-based design is an iterative and collaborative process involving designers and analysts, spanning all PDP phases and targeting the optimisation of design-simulation loops in the different PDP phases [Bajaj et al., 2007].

Figure 11 shows a design-simulation loop process as it can be observed at Snecma in disciplines using computer-aided design (CAD) models as input for their calculations. In that case we illustrate a mechanical analysis performed on a compressor disk.

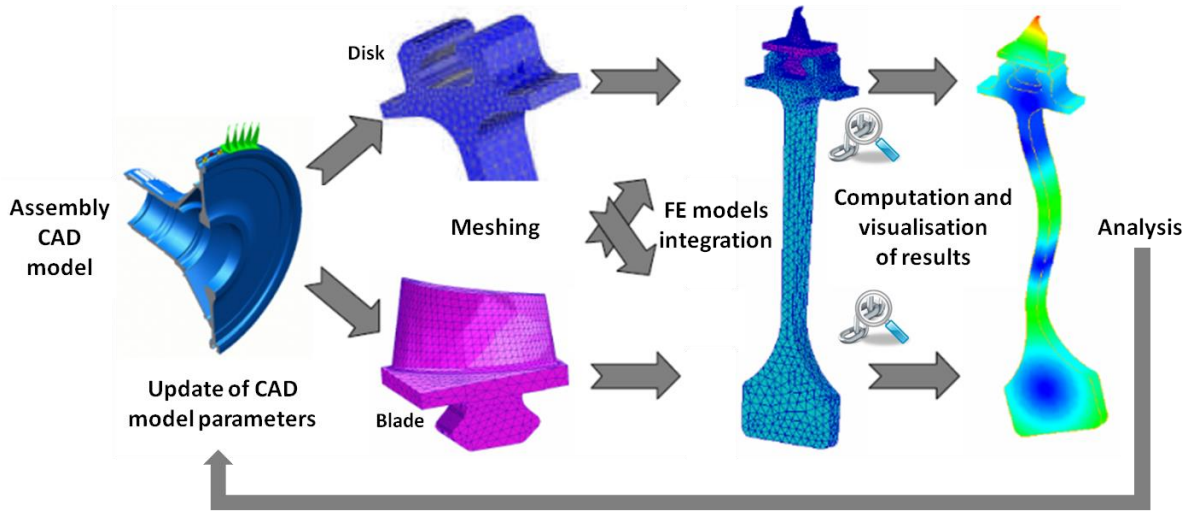


Figure 11: Mechanical Design-Simulation loop of a compressor disk

To complete this figure, Figure 12 displays the various data involved during such a loop as well as the relationships between CAD and CAE activities and data. It underlines that a numerical simulation involves a lot of input data and requires a significant modelling work. Indeed, it is about describing the

most relevant physical data while taking into account the limitations of the modelling and computational constraints. The relationship between design and simulation appears to be a crucial iterative activity.

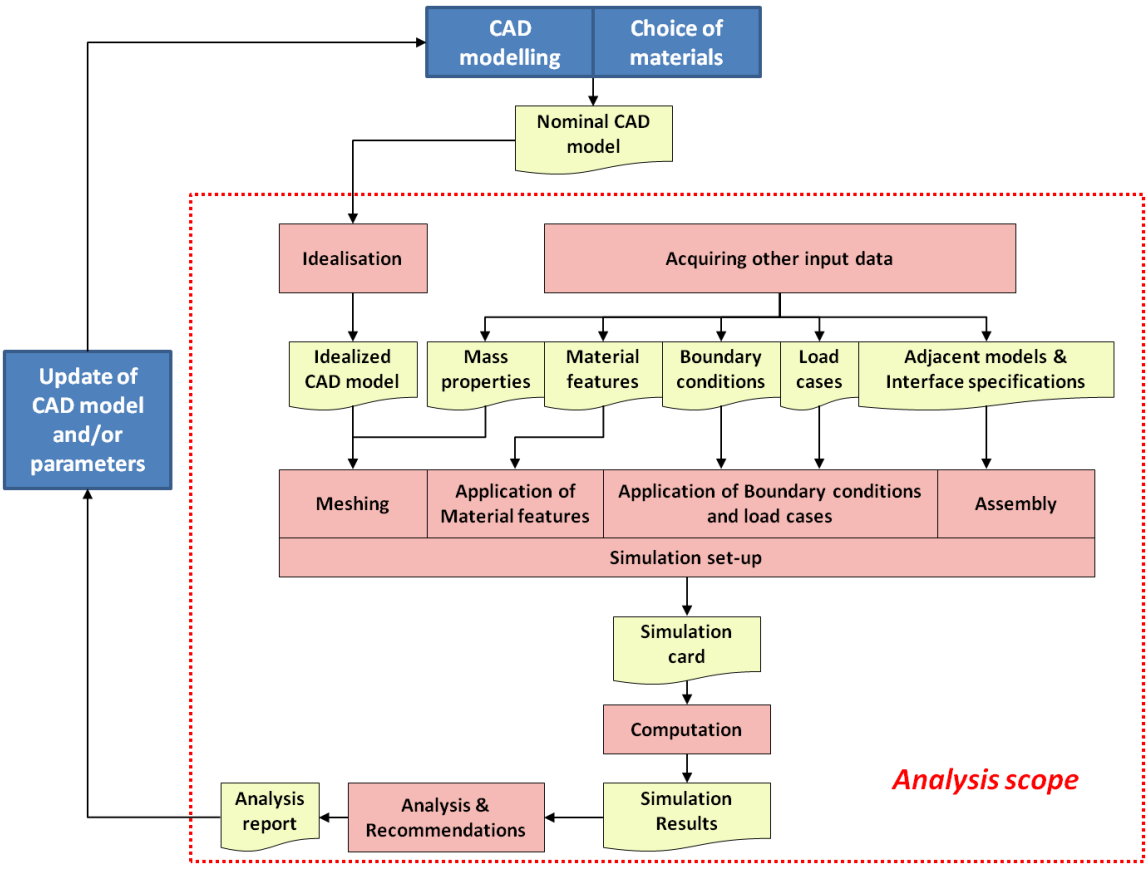


Figure 12: Design and Analysis data involved in a CAD-CAE loop process

As illustrated on Figure 13, Bajaj proposes to formalize the scope of SBD with the Gero's function–behaviour–structure (FBS) framework.

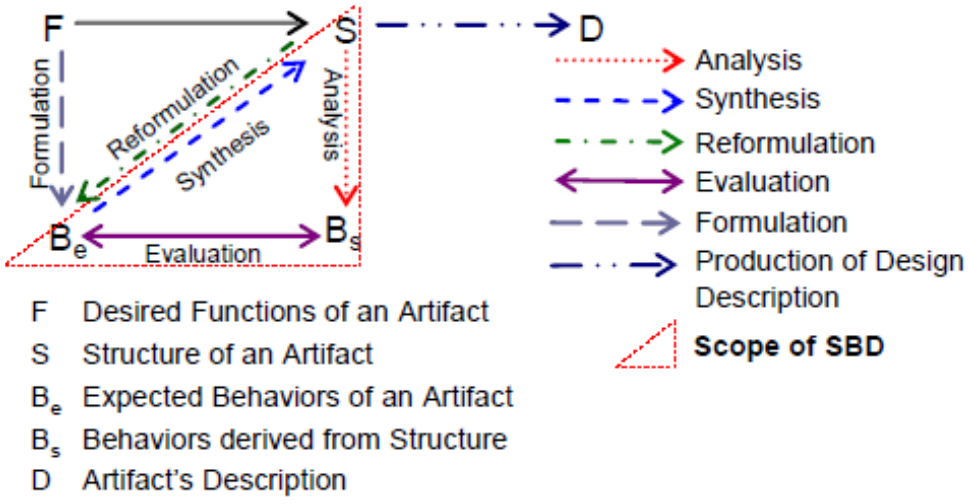


Figure 13: The FBS framework and scope of Simulation-Based Design (from [Bajaj, 2008])

The basis for Gero's FBS framework is formed by three classes of variables describing different aspects of the product design object [Gero&Kannengiesser, 2004]:

- Function (F) variables: describe the teleology of the object, i.e. what it is for.

- Behaviour (B) variables: describe the attributes that are derived or expected to be derived from the structure (S) variables of the object, i.e. what it does.
- Structure (S) variables: describe the components of the object and their relationships, i.e. what it is.

A designer constructs connections between the function, behaviour and structure of a design object through experience, knowledge and know-how. Specifically, the designer ascribes function to behaviour and derives behaviour from structure. As shown on Figure 13, Bajaj [Bajaj, 2008] makes clearly the analogy between the scope of simulation-based design and the FBS framework design process [Gero, 1990]. The scope of SBD corresponds exactly to the scope of design-analysis iterations. The design part consists in defining and specifying components and interfaces (the structure S) and the analysis part consists in first defining the expected behaviour (Be) based on functional requirements (formulation of simulation objectives) and in analysing the behaviour of this structure (evaluation permitting to get the actual behaviour (Bs) of the structure). The analyst compares the simulation results to the functional requirements that the simulation must verify (synthesis). Then, the design structure can be whether validated whether re-designed leading to another design-analysis iteration where whether the structure whether just some design parameters are modified (reformulation) and the simulation models updated for a new analysis and synthesis.

However, one major challenge in this FBS triptych is that functional and design information has to be suitable to the product operational state, its configuration within the design process and the discipline of the simulation. The analysis process consisting in deriving Behaviours (Bs) from Structures cannot be achieved directly because it requires defining domain-specific behavioural configurations. This is one of the major challenges of simulation-based design: managing the system/product complexity by integrating and coupling various functional variables with different multi-level and multi-domain structures and behaviours.

2.4 Conclusion

Developing complex aeronautic systems such as aero-engines in the current business context (faster, better, cheaper) require having a precise view of the structural and multi-disciplinary system complexity in order to better organise and synchronise design and simulation activities between co-designers and partners within the extended enterprise.

Complex PDP is always subjected to risks, uncertainties, as well as difficulties to coordinate inter-disciplinary design activities. System Engineering and Integration methodologies and standards provide a process framework to handle this complexity and these uncertainties. All along an IPPS development life cycle, the tight coupling of several phenomena interacting together makes it a great integration challenge to find a global optimum for the entire system. The search for this optimal performance stresses the need to adopt an **integrated and simulation-based design approach.**

The synchronisation and the reduction of time and knowledge gaps between multi-domain and multi-level interdependent simulations, performed by different partners using different tools, are major integration challenges. These gaps and uncertainties can be addressed by fastening the design-analysis iterations. The design-analysis iterations and integration challenges have been formalized with

the help of the FBS framework. This has enabled us to underline the need to provide appropriate "definition view"; characterising the ideal content and organization of product data (corresponding to the appropriate behavioural structure) for a specific discipline and simulation context.

System engineering standards and methods only describe each process task outcome and do not specify the details of "how to" implement these processes for engineering a system. Neither do they specify the methods or tools the designers/integrators could or should use to implement the SE process, nor do they prescribe the name, format, content and structure, of the design product data. The collaborative and multi-disciplinary nature of aeronautics development programs as well as the need for integrating a great number of product components as well as related models often results in large quantities of intermediate design data with heterogeneous formats and complex relationships. The efficient organization and management of engineering data are therefore a bottleneck of product design performance [Feng et al., 2009]. Therefore, adequate collaborative design environments are necessary to ensure that partners and co-design teams can share or/and exchange engineering data created all along the product development life cycle [Kleiner et al., 2003] [Eynard&Yan, 2008]. These collaborative digital environments might provide an **integrated IT design environments enabling partners involved in the extended enterprise to harmonize their design processes exchanging the appropriate data in the appropriate context.** The objective of defining an IT integrated design environment is to provide an **integrated view of the product** namely a product reference framework for the project partners. The key resulting issue is how to manage and enable the data migration between partners' IT environments systems, and how these exchanges would be managed through the integrated design environment [Kleiner et al., 2003]. This integration of data consistency is strongly linked to the concept of **interoperability** which can be considered as the environment capability to enable multiple systems and multiple partners to access data in a multi-view design approach.

Therefore, the remaining field to analyse to fully identify all the industrial challenges and issues is the field of digital design environments and tools. Next section aims at characterizing the main digital engineering capabilities used in current PDP activities as well as to highlight the remaining challenges and required capabilities to handle complex system design and integration more efficiently.

Chapter 3: Collaborative digital design environments

The development of complex systems like an aero-engine, requires the implementation and the use of methodologies and tools (supporting the methodologies) enabling to manage efficiently the product complexity. Aeronautics industry and more especially PDP activities have been impacted in recent years by the advent of the use of digital engineering such as CAX (computer-aided for X) systems. According to the various application fields, different **CAX-systems** were developed: **Computer Aided Design** (CAD) for product design; **Computer Aided Engineering** (CAE) for validation of the product definition/design (validation on virtual product behaviour); the **Digital Mock-Up** (DMU) enabling collaboration and contextual design. On one hand, these tools can be seen as supporting the management of the product and organisational complexity introduced in previous chapter. In the other hand, the enhanced use of CAX technologies has increased the amount of data created during the PDP. It has also created a big potential for value creation and conversely, information and knowledge waste. Enabling movement of such masses of data through the different activities has enhanced the need for **Engineering Data Management** (EDM) systems. Their use is nowadays inseparable from CAX systems. All these digital engineering environments are supporting what we have called the **digital integration chains**; a digital design process that is introduced in this chapter.

3.1 CAX systems

3.1.1 Computer Aided Design

Computer Aided Design (CAD) consists in using computer systems to assist engineers in the creation, modification and optimisation of a design [Narayan, 2008]. CAD tools enable to make design changes at lower cost and have 2D and 3D representation in the space of a part or an assembly. At Snecma designers use CATIA V5 which is recognized as one of the most complete suite of CAD tools (see opposite Figure 14).

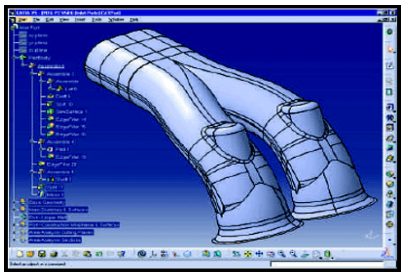


Figure 14: CAD model created with Catia V5

Today CAD modellers have integrated new functionalities to meet the need of reducing design cycle times among which feature-based design, assembly design, kinematics, material application, etc. [Fuh&Li, 2005] [Li et al., 2005]. **Featured-based design** and modelling have become the most common method used by mechanical engineers [Li et al., 2004]. It consists in storing in CAD models not only geometric information but more function-oriented information and design intents. This is done using features in feature-based models. Therefore, feature-based modellers allow operations such as creating holes, fillets, chamfers, bosses, and pockets to be associated with specific edges and faces.

Another important CAD capability is the ability to link and assemble different CAD models. This associativity allows models to implement, among other things, a so-called **design context method**. The design context facilitates updates of assemblies. It also prevents that neighbouring parts interpenetrate.

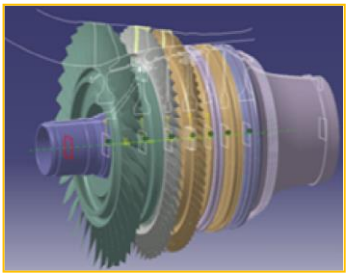


Figure 15: Example of design in context

This approach relies on two axes:

- Using interfaces;
- Use of adjacent parts being designed.

CAD interfaces are a particular category of 3D objects. They define the links between two 3D models [Nguyen Van, 2006] . In contextual design, interfaces permit to define the limits and the positioning of an object in its environment; i.e. its space allocation. In Figure 16, it is explained how a design based on an interface definition allows a quick update of its position within an assembly.

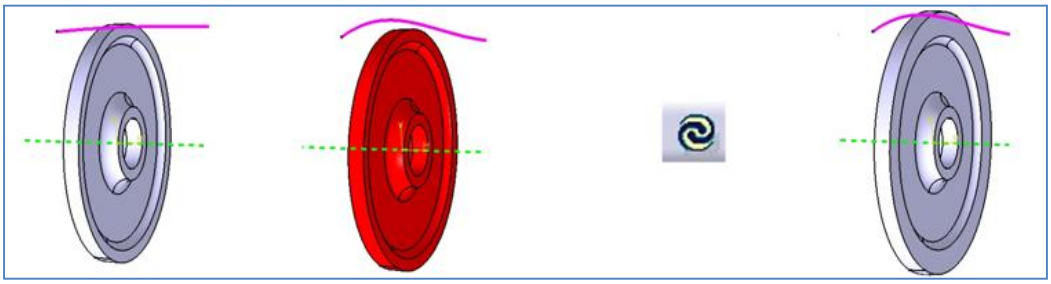


Figure 16: Example of updating a disk which contextual design is based on an aerodynamic outline

However, although today CAD systems allow designers to store a great amount of useful product and engineering knowledge and data into the CAD models, there is a crucial lack of modelling standardisation (especially in large scale projects where the product consists of thousands parts designed by distributed design teams). This standardisation can concern model format, the modelling methods or the content and structure of information contained in the CAD models. CAD is only one part of the digital product development activity. Within the product lifecycle CAD applications are not used as stand-alone applications. CAD tools and models are the basis for digital product definition and are necessarily coupled with the use of other digital engineering applications such as Computer-aided engineering (CAE) such as Finite element analysis (FEA), Computer-aided manufacturing (CAM) and Engineering Data Management systems.

3.1.2 Computer Aided Engineering and Finite Element Analysis

Computer-Aided Engineering (CAE) systems are dedicated to the verification and validation of the product design definition enabling to perform deep analysis, simulation and validation of virtual product behaviour. Nowadays CAE applications encompass many engineering disciplines among which:

- Mechanical analysis: stress analysis (dynamic or static, linear or non-linear) and mechanical vibration using Finite Element Analysis (FEA);
- Thermal and fluid flow analysis Computational fluid dynamics (CFD); Heat transfer (conduction, convection, radiation);
- Performance simulations;
- Multi-body dynamics and Kinematics;
- Acoustics;
- Manufacturing analysis, process analysis;

Finite Element Analysis (FEA), as applied in engineering, is a computational tool for simulating the behaviour of an object using the finite element method. This method consists in breaking a real object down into a large number of “finite elements” (e.g. cubes, tetrahedrons, octahedrons, etc.) that constitute the Finite Element Model (FEM) generated through the use of mesh generation techniques and

FEM algorithms. The behaviour of each finite element is predicted by a set of mathematical equations representing physical phenomena laws (e.g. Euler-Bernoulli beam equation, the Navier-Stokes equations). The sum of the individual element behaviours produces the expected behaviour of the whole studied object.

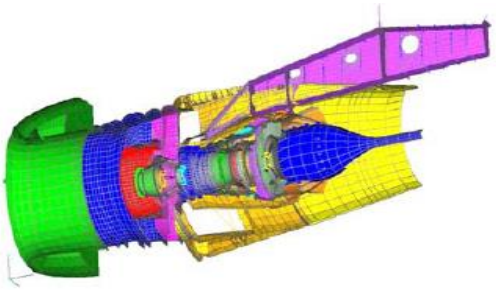


Figure 17: Integrated Mechanical Finite Element Model of a power plant system

In general, there are three phases in any computer-aided engineering and more particularly in a FEA process:

- **Pre-processing**: that consists in creating the FEM and defining the environmental factors to be applied to it. Therefore, FEA begins with the use of the finite-element modeller (sometimes called a mesher or pre-processor).
- **Analysis solver** (usually performed on high powered computers): Solvers are the engines of FEA. They take elements, boundary conditions, and loads in order to output a solution containing all the information needed to review and understand results. Solvers may be divided into two categories: linear and nonlinear.
- **Post-processing** of results (using visualization tools): utilize the data generated by the solver to create easily understandable graphics and reports (see Figure 18 below).

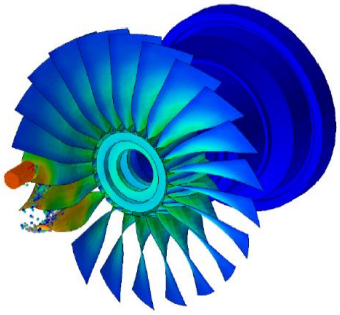


Figure 18: FAN blade-off simulation results due to a bird ingestion

It is important to notice that the time to perform a FEA process and resulting cost are heavily dependent on the pre-processing phase since the most consuming time activity is the creation of the model for analysis [Zhou et al., 1997]. That is why, in order to effectively incorporate analysis into the design cycle, this phase consisting in creating the required models for analysis need to be the most efficient possible. In order to generate the appropriate mesh for a specific analysis, FE modellers require CAD data inputs and physical properties about the modelled components. Much of the time and effort of creating analysis models is a result of not using the information from CAD design models or past analysis models [Mocko&Fenves, 2003]. Moreover, acquiring the data inputs (that can come from various data sources) to create an integrated FE models is a very time-consuming and tedious work.

3.1.3 Digital integration chains and the issue of FE assembly models

A **digital integration chain** is a process where subsystems (or components) models are integrated enabling system behaviour prediction, validating expected system performances.

In order to analyse problems in digital integration chains we have conducted and observation on the mechanical integration process of an IPPS. In Figure 19, we have schematised this IPPS mechanical digital integration chain. In this example, the integration is performed in a collaborative and distributed environment between the integrator of the IPPS and the integrator of the engine sub-system.

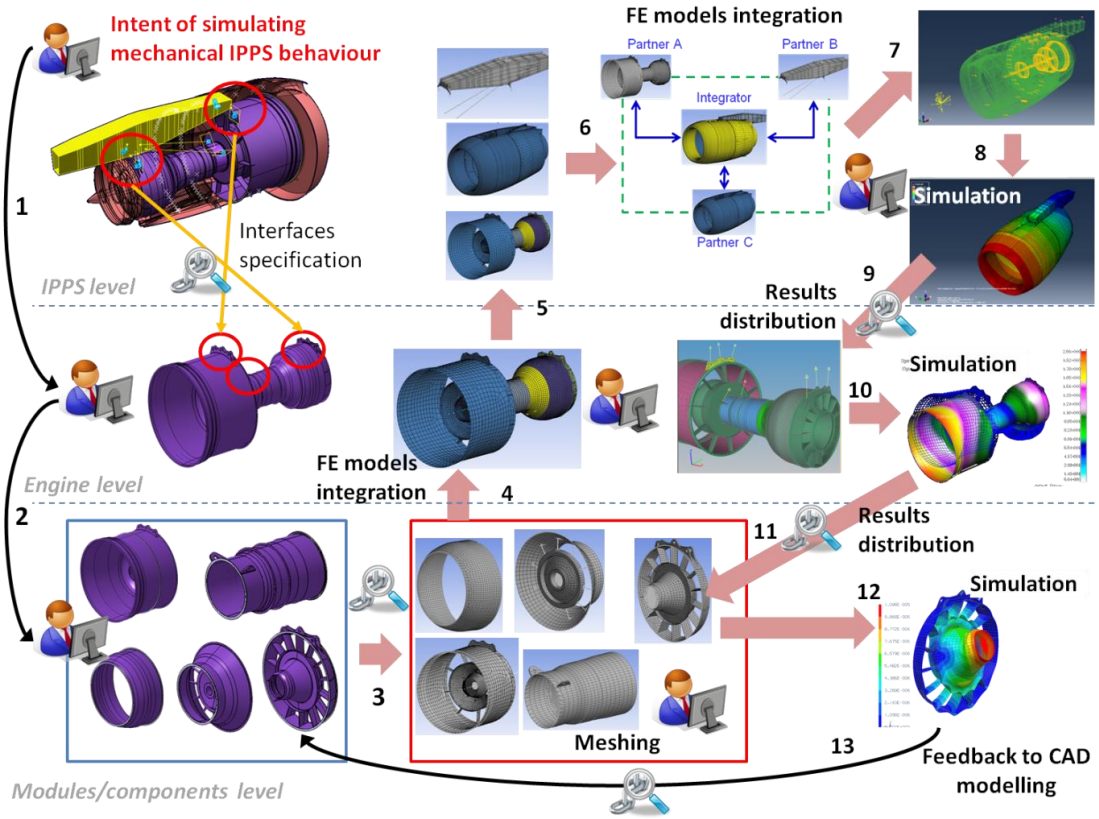


Figure 19: Part of a mechanical digital integration chain at IPPS and engine levels

This process occurs at each system level, starting by defining and cascading simulation objectives and requirements at system level, defining the interfaces between the components (1-2), designing and meshing the sub-systems (3), integrating these models (4-6), setting-up and performing the simulation (7-8) and distributing results for downstream simulations; whether for other disciplines, whether for simulations at lower system breakdown distributing specific results at interfaces as shown in Figure 19 (9-12). If the simulations performed within these digital chains do not validate the expected related behaviour, they also include feedbacks from simulation results to CAD design and the whole change impact chain (e.g. 13); leading to new design-analysis iterations

Unlike modelling a standalone component having no adjacent component, a mechanical assembly model must be able to transmit displacements/stresses from one component to another. An assembly simulation model is not just a set of meshed sub domains positioned geometrically in a global coordinate system. These sub domains must be interconnected to each other to generate global displacements and stresses. Therefore, the preparation of an assembly model compared to a standalone component implies a need to clearly define and prepare interfaces specifications.

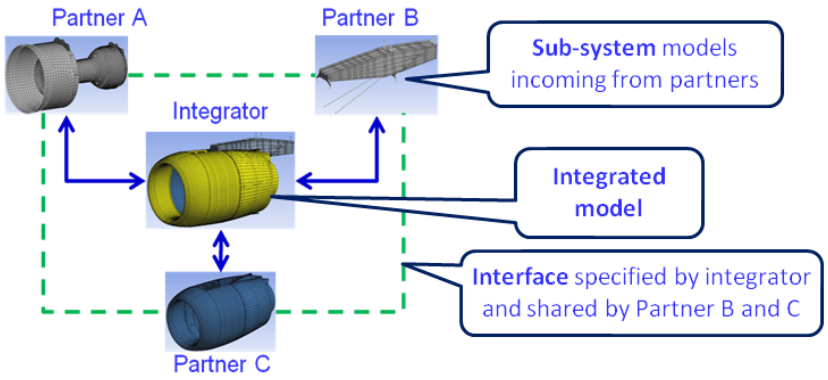


Figure 20: IPPS mechanical model integration

An assembly simulation model derives from shape transformations to produce a mechanical model containing a set of domains discretised into FEs connected together. From this perspective, idealization eases the integration of simulations in PDP. However, even if the idealized sub domains reduce the analysis time, today they are obtained through a very tedious preparation process. To process large assemblies with over hundreds of parts, an automation of this preparation process is required to better and faster integrate assembly simulations within a PDP. To speed-up this process, it is necessary to identify the origin of lengthy operations.

Moreover, depending on the discipline and the type of analysis performed, these digital integration chains and related numerical simulations require defining specific product architecture models (using specific representation of the product) in order to create the appropriate simulation models. A major issue for the integrator is to manage these models so as to identify the relevant data set to be used for the simulation and to organize this data set into a new adapted product structure and "engineering environment". Therefore, in order to ensure the continuity of information within digital integration chains and between involved design working teams, we can state several challenge in complex product design:

- **Interoperability between systems**: from our observation, we noticed that there was still a lack in integrating efficiently tools and then data conveyed through these tools. This lack of efficiency is currently a problem while attempting to communicate in a meaningful way between heterogeneous CAX and EDM systems; which explains the importance the importance of implementing product data standards in such applications.
- **Consistency between data and models**: a consequence of interoperability constraints (even using standards) is the loss of data consistency between data managed in different domain-specific tools but also between tools of the same nature. The model consistency concerns the quality and the semantic of the data conveyed through these models. This loss of consistency causes the problem of data interpretation between multi-partner and multi-disciplinary co-designers.
- The **control of design and analysis data through an integrated reference framework**: this environment is required for a better integration and traceability between design and analysis data in view to provide the appropriate "analysis product views" and enhance re-use of appropriate design artefacts (e.g. CAD models, physical properties or past analysis models) regarding the scope and objectives of the various performed analyses.

These represent major challenges in complex product design: make Engineering Data Management systems and Digital Mock-Up functionalities in phase with these digital integration chains needs.

3.2 Introduction to Engineering Data Management Systems

Engineering Data Management (EDM): is the process of organizing, structuring, storing, and tracking the design information created by engineering and development activities [Feng et al., 2009].

Complex product design process is tentative and iterative. Therefore it produces large scale intermediate data with heterogeneous formats and complex relationships. The use of EDM systems is essential for PDP performance (enhancing information and communication flows) and is inseparable from CAX systems. This approach coupled with the need to manage and access different data created all along the lifecycle (for the use of connected activities such as configuration management or manufacturing) increases the need for EDM systems deployments.

Product Data Management (PDM): PDM systems aim at managing and storing the product data and also information related to its entire lifecycle (design, manufacturing, assembly, maintenance, etc.) [Eynard et al., 2006]. A PDM system provides computational tools that support the management of either engineering (or product data) and development process information (or product workflow) [Feng et al., 2009] [Stark, 2011].

These tools provide valuable functionality with process management particularly as it relates to configuration management or engineering change control [Crow, 2002]. PDM systems typically manage product-related information such as geometry, engineering drawings, project plans, product specifications, analysis results, bills of material, engineering change orders, and many more. PDM can also be seen as an integration tool connecting many different areas of product development, which ensures that the right information is available at the right moment for the right actor and in the right format with the right semantic objects for activity [Chen&Jan, 2000] [William Xu&Liu, 2003]. In other words, PDM should manage consistent and meaningful information regarding the lifecycle stage and the activity, in order to create the most adapted environment for the engineer. This leads to the concept of Product Life Cycle Management (PLM).

Product Lifecycle Management (PLM): PLM is the activity of managing company's products all the way across their lifecycle in the most effective way. It aims at bringing better product to market faster [Stark, 2011]. PLM is a business strategy that helps companies share product data, apply common processes, and leverage corporate knowledge for the development of products from design to retirement, across the extended enterprise [DS, 2010].

PLM actually has its inception in PDM applications which are the major and most important component of PLM systems. PLM solutions help to define, execute, measure and manage key product-related business processes. It should links information from many different authoring tools and other systems to the evolving product configuration. More than a tool, PLM is then a strategy for integrating, tools, actor and processes within a common referential environment enabling to manage, share and exchange product data all along the product life cycle. . Now, with the multiplication of CAE applications and with the deployment of simulation-based design approaches in the aeronautics industry, the new trend in PLM systems is to integrate and manage simulation processes and related data enlarging its functional scope with Simulation Data Management and Simulation Lifecycle Management systems.

Simulation Lifecycle Management (SLM): Lalor defines SLM as the management of the intellectual property associated with simulation tools, data, and processes as related to product or process development [Lalor, 2007].

To bring simulation data and processes into the enterprise lifecycle, SLM must be considered as a major part of the PLM strategy and systems. As traditional PLM routinely captures the form and fit of product designs through digital mock-up (DMU), SLM compliments PLM by associating behavioural simulation data and processes with the DMU [CIMdata, 2011]. In doing so, the objective is to provide a single source of "truth" for all design and analyses information and processes. Therefore, a major objective of SLM is to transform simulation from a specialty operation to an enterprise product development enabler that spans many segments of the product lifecycle. To do this, SLM should provide technology in four foundational areas [CIMdata, 2011]: Simulation and test data management (and their correlations), Simulation and test process management, Decision support and Collaboration.

Simulation Data Management (SDM): SDM systems are as PDM for PLM the fundamental basis for SLM. This is a specific PDM system which manage all the necessary engineering data (CAD inputs data, loads, boundary conditions, material properties, scenario of simulation, results reports, etc.) permitting the engineers to perform simulations in the most efficient way.

PDM and SDM systems are built differently since SDM are not built around the product structure but use product data and organize links in a specific way and with specific rules driven by simulation processes and data. Simulation data management is similar functionally to PDM but designed specifically for analysis needs. It enables users to more easily and quickly find the simulation information they need. They can query for simulation inputs and results and the status of simulation processes.

As already mentioned and as shown on Figure 21, PLM had its inception in PDM applications during the mid 1980s. In the last two decades, the increasing amount of data to manage through these systems and the increasing collaborative needs of the extended enterprise have made the scope and capabilities of EDM systems evolved so as to support the collaborative creation, management, dissemination, and use of product definition information.

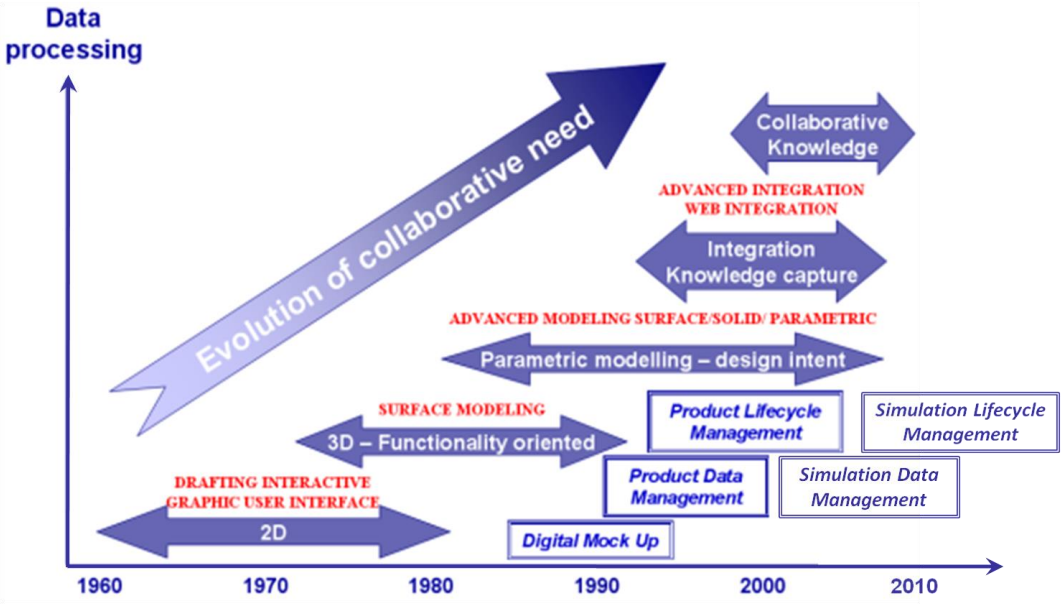


Figure 21: Evolution of digital engineering tools functionalities – adapted from [Nguyen Van et al., 2006b]

Among all these CAX and EDM applications, the Digital Mock-Up has become the major federating environment of the product definition. It provides an integrated view of the product where geometric data and other aspects of product definition can be linked to the configuration management aspects (bill of materials, product configurations, engineering change management process, etc.).

3.3 The Digital Mock-Up: a major industrial stake

3.3.1 Digital Mock-Up's fundamentals

The digital mock-up is a collection of CAD 3D models which are positioned in 3D space to represent the form of the product to be developed. The DMU enables collaboration and contextual design which permits different partners to delimit their 3D working environment through an integrated view of the models [Sadoul, 2000] [Eynard&Yan, 2008]. It is as a major tool supporting concurrent engineering [D'Adderio, 2001] and enabling limiting physical prototyping.

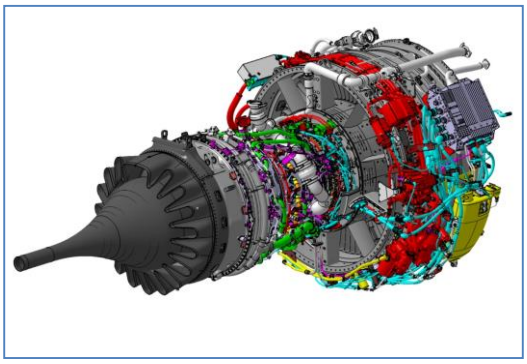


Figure 22: SAM146 digital mock-up

A DMU is created through the integration of a CAD system with a PDM system (see Figure 23 below). All geometry is accessed via a database. Models may be in database or pointed to by the database. Each model is attached to a part in the Bill Of Materials (BOM) and the product data structure of the PDM. The links between models and product data are defined in the mock-up data database.

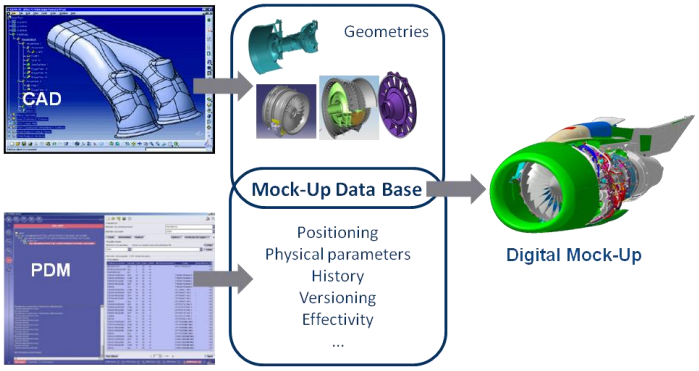


Figure 23 : Digital Mock-Up schema

3.3.2 Digital Mock-Up Applications

One of the most important functionality and the original reason to which DMU have been implemented in industries is that it allows **collaborative contextual design**. Indeed, the DMU provides a common framework for all partners to position the modules. To this end, CAD files for interfaces are created and positioned in the DMU. Like the CAD interfaces introduced in section 3.1.1, the CAD definition of partners' modules are based on these interfaces. Thus the modules are assembled correctly and parts updates is faciltated when interfaces are modified.

The second advantage of the DMU is that it integrates partners CAD data, which allows designers to use these parts as the context of their own design, preventing from having parts interference/clashes. The DMU provides an overall vision for verifying the assembly design of the engine. Indeed, it enhance error detections earlier in process performing several analyzes of the complete engine assembly model:

- positioning analysis: detects interference/clashes between parts;
- kinematic analysis: to determine whether the rotor parts do not fit in stator parts;
- analysis of assembly / disassembly: simulates the operations of assembly and disassembly of parts to optimize the number of operations required for these tasks
- accesibility analysis: checking accessibility of components for maintenance.

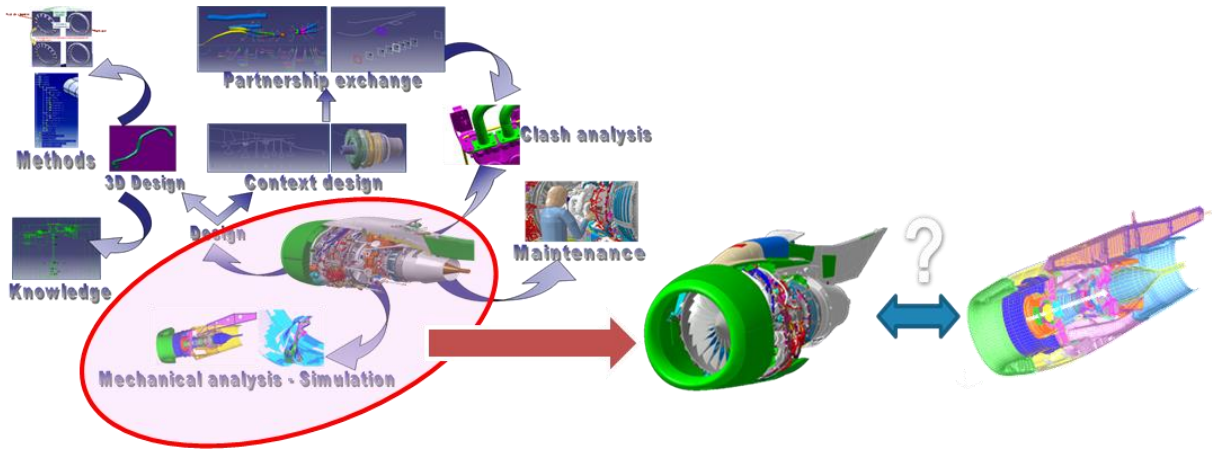


Figure 24: DMU applications adapted from [Nguyen Van, 2006]

In [Nguyen Van, 2006] and as shown in Figure 24, the author has identified the different potential usage of the DMU. However, the high complexity of aeronautics products containing thousands of objects performing many different functions requires DMU adaption and transformations to different user's needs and knowledge.

DMUs can also be used by analysts to approach more complex geometry and speed up simulation models generation. In order to reach the needs of large assembly simulation models it is mandatory to speed up as much as possible the required DMU transformations to meet simulation objectives [Boussuge et al., 2012]. One of the objectives of this PhD dissertation is to characterize and analyze some specific issues related to the use of DMUs for assembly simulation model preparation.

A DMU is actually an extraction of a given configuration (defined in a PDM system), at a given time for a given use. This is done via a request to the PDM system that returns a product configuration. When the DMU is used, the DMU is not any more configured by the PDM, but corresponds to a specific technical solution, representing the geometry of the product components positioned in the reference frame of the product. It can therefore be assimilated to a snap shot at a given time, i.e. at a maturity level of the product configuration definition.

3.3.3 Evolution of a Digital Mock-Up during the product life cycle

As shown on Figure 25, the construction of the DMU starts from the earliest design stages. Figure 25 exemplifies the mechanical skeleton of an aero-engine which is the starting point for the engine DMU defining the different space allocations required for the design of the sub-systems and components constituting the power plant system. In the case of an aero-engine, these space allocations are

specified by the aerodynamics outlines, the surface of the blades of each modules and the interfaces' plans and design patterns permitting to position and size them.

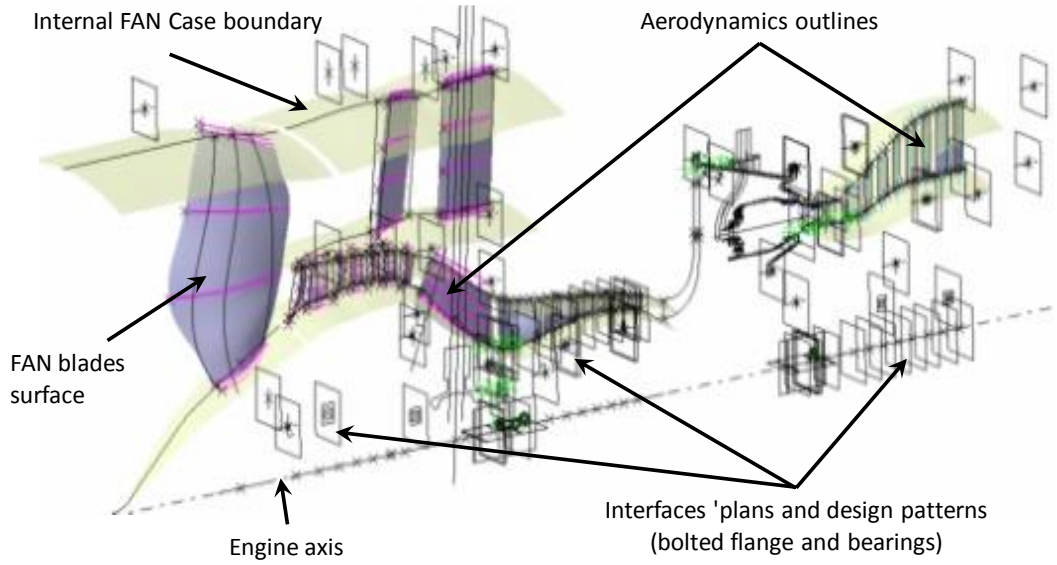


Figure 25 : Design in context in preliminary design – Mechanical Skeleton of an integrated aero-engine assembly

From the view of the mechanical skeleton illustrated above, the DMU evolves during PDP toward more and more detailed representations since it is progressively enriched with the detailed definition of the product components. As observed in the context of Snecma, Figure 26 describes the evolution of an aircraft engine's DMU all along its development life cycle. In the preliminary stages, the geometrical representation is based on the 2D cross-section and then on the mechanical skeleton of the product. Once the concept is validated, the cross section is transformed into a 3D model using rotation. At this stage, the DMU represents the design space allocated for the different components of the product. In the definition stage, the DMU is deeply modified. As design departments are working on the development of modules, the new design definitions are integrated into the DMU to provide a new realistic representation of the product. Finally, before the service stage, the DMU is the final virtual product which will be transformed into a real product after manufacturing.

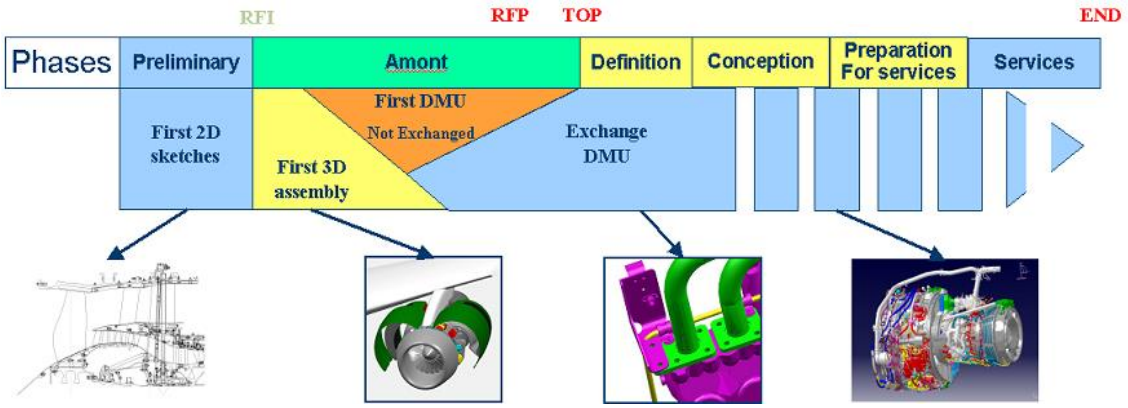


Figure 26 : Parallelism between project stages and DMU evolution [Nguyen Van, 2006]

3.3.4 Synchronizing the DMU with the product definition

The DMU and its evolutions deeply depend on the evolution of technical definition by design offices. Furthermore, all modifications performed during the design definition must be integrated into the DMU. In consequence, all internal workflows and procedures must be adapted so that the DMU

evolution is consistent with the current state of the technical definition. It means that all information concerning the design technical definition must be integrate into the DMU (3D models referencing validated drawings, reference, “effectivities”, quantity of items, functional item relationships, etc.). The technical definition of an aero-engine is driven by the configuration management process.

One of the major processes supporting and allowing management of design definitions and their evolutions is the configuration management process.

Configuration management (CM) is a process enabling to establish and maintain consistency of product’s performance, functional, and physical attributes with its requirements, design and operational information throughout its life [DoD, 2001].

CM emphasizes the functional relation between parts, subsystems, and systems for effectively controlling system change. It helps to verify that proposed changes are systematically considered to minimize adverse effects. CM verifies that changes are carried out as prescribed and that documentation of items and systems reflects their true configuration.

Configuration: Rule describing the composition and structure of a hierarchical system. Configurations are used to describe different product structures adapted to the needs of the project.

The definition of the components (item instances) that constitute a configuration is managed through the concept of “effectivity”.

Effectivity: An effectivity is the identification of a domain of applicability for product data [ISO, 2004a]. From a PDM/PLM perspective, the effectivity is a product structure link attribute that defines if a usage occurrence of an item definition (i.e. instance of component) is relevant for a given configuration or list of configurations.

Our field research has shown that the product configurations are defined at Snecma according to the product variants defined for specific customers needs and according to the various “engine-mounts” couples (called units) defined by the test plan. The configuration management process encompasses:

- The creation of **items** and technical definition that define and compose the product structure;
- The definition and the management of the **Bill of Materials** (BOM): that specifies the parts that compose a product configuration. The parts’ attributes generally are: the functional reference (SNS), the part number, the designation, the mass parameters (masses, inertia, centre of mass), the material, the quantity;
- The **engineering change management process**: a review process ensuring that for each modification or revision of items’ definition, changes to the system are proposed, evaluated, and implemented using a standardized, systematic approach that ensures consistency. Proposed changes are evaluated in terms of their anticipated impact on the entire system.
- The **synchronization with partners’** modules definition and the **specifications of modules interfaces**: this process especially specifies interfaces (both they provide to the appropriate actors **interface control drawings** that specify functionally and physically the interfaces.

As a result, the content and structure of a DMU and its evolution is closely related to the configuration management process. The bill of materials is traditionally considered as the reference of the technical definition. The evolution of the DMU actually follows the technical definition defined in the

BOM. Each part has to be associated to the appropriate product configuration(s), and has to be positioned into the integrated environment of the DMU. As an objective, the Digital Mock-Up data must reflect exactly all the data (means reference, SNS, quantity…) that are present in the BOM. The coherency between DMU and configuration management must be ensured to provide the best design environment and view on the product. To summarize the quality of a DMU can be measured assessing the following criteria:

- **Completeness**: number of existing parts regarding the parts referenced in the BOM, presence of CAD models (internal or partners' models), availability of models at the right time, synchronization with partners' DMU, presence of partners' models.
- **Number of clashes or gaps**: due to a bad positioning of the models in the relative product referential or due to discrepancies between the definition of the models to assemble.
- **Number of incoherencies with the BOM**: when the parts present in the DMU don't correspond to the parts referenced in the BOM;
- Number of discrepancies between the whole product 2D sketches and the DMU.

An aero-engine DMU, composed of thousands of objects and built from the collaboration of hundreds or thousands designers, require to be enriched correctly and to match the current product definition. As a result, with such an approach (considering the BOM as the product definition reference), there is always a gap between the current product definition and the contents of the DMU. In a model-based approach (see section 6.1), the opposite approach must be used: the DMU must be the reference of the product definition and the BOM must be generated from the DMU. Doing so, inconsistencies between these two product definitions are avoided. However this requires establishing very strict and standardized rules and methods between all co-designers and partners involved in the enrichment and exploitation of the DMU. For instance, the content of CAD models that enrich the DMU must be rich enough to produce relevant BOMs. For a large and complex product developed in the context of the extended enterprise (distributed and collaborative environments), where partners might use different CAD and PDM systems, as well as different routines and methods, this appears as one of the major challenges and industrial stakes that these companies are facing today while managing DMU environments.

3.4 Conclusion: From Digital Mock-Up to Behavioural Mock-Up

The development and use of DMUs in a Product Development Process (PDP), even with large assembly models, make 3D CAD models available for the engineers. The DMU offers new perspectives for analysts to approach more complex geometry and speed up the simulation model generation [Boussuge et al., 2012]. In view to this perspective and in view to extend the scope of PLM adding SLM functionalities, **another big DMU-related challenge for these companies, is to integrate behavioural simulation data and processes with the DMU;** offering what Dassault Systèmes and CIMdata have called **Behavioural-Digital Mock-Up**.

The **Behavioural Mock-Up** (BMU) concept was initiated by Riel who extends the DMU concept with the required information using a meta-modelling approach [Riel, 2005]. The BMU is the equivalent of the DMU for simulation data and processes. Beyond the geometry, which is represented in the DMU, the so-called BMU should logically link all data and models that are required to simulate the physical behaviour and properties of a single component or an assembly of components. These include mechanics, thermal, hydraulics, electronics, and whatever other disciplines which are relevant. Probably the most fundamental requirement to the BMU is its ability to be integrated into existing IT infrastructures. Since the BMU aims at acting as a layer above all the different modelling tools, it will have

to have interfaces to all of them. This can be a major problem in an area where there are no interface standards available (like in CAE) [Riel, 2005].

The key enabler for achieving the target of extending the concept of the established CAD-based DMU to the behavioural CAE-based BMU is to find a bi-directional interfacing concept between the BMU and its associated DMU. The BMU could use all the DMU-data it needs for model calculation, and all relevant BMU calculation results could be made available and physically accessed via the DMU. Within this relationship, the DMU could serve as the key link between the BMU and the PDM-system. These requirements demand a concept to handle the structural information that is necessary to determine the relationships among components and to map DMU and BMU content and structures [Riel, 2005]. Riel did not define precisely neither demonstrate this concept in his PhD, but it is now the aim of SLM systems and of some R&D aeronautics consortium projects.

This is as well one of the main objective of this PhD which is realized within one of those R&D projects: the CRESCENDO project. CRESCENDO means **C**ollaborative & **R**obust **E**ngineering using **S**imulation **C**apability **E**nabling **N**ext **D**esign **O**ptimization. This European consortium involves 59 partners representing a cross section of European aeronautics. This includes European aircraft, engine and equipment manufacturers, some universities, research institutes and PLM/SLM and CAD/CAE software vendors. The project aims at delivering the modelling and simulation backbone of the aeronautical extended enterprise: the **Behavioural Digital Aircraft** (BDA). The BDA concept represents the BMU of the aeronautics extended enterprise and might consist in a collaborative data exchange/sharing platform for simulation processes and models throughout the development life cycle at aircraft level and in the entire supply chain. The BDA is the system which will describe and host the set of simulation models and processes enabling to link, federate, couple and interact with different models with seamless interoperability, hierarchical, cross-functional and contextual associativity [CRESCENDO, 2010].

Regarding the diagnosis of the current industrial context, related issues and requirements, we have highlighted and enhance the need of harmonization between design and simulation activities and their related data and technologies. The primary questions that this research intends to answer can be formulated as below:

In the context of a collaborative, distributed, multi-partners, multi-level, multi-physics and multi-domain design environment, how to align digital engineering capabilities with business processes and specific requirements associated to product/system integration?

In this environment, what are the appropriate methods and tools to provide the right product engineering data (with the necessary level of information and in the right format) in the appropriate engineering context, to the right person and at the right moment?

Regarding the importance and the needs of design-simulation loops and integration activities in SBD process, we have focused this general issue in a more specific and model-based oriented issue derived from the perspectives of Riel's PhD:

How to make DMU the key link between the BMU and the PDM-system? How to handle the structural information that is necessary to determine the relationships among components and to map DMU and BMU content and structures?

Chapter 4: Key technical requirements

The industrial context and related concepts described in this chapter have permitted to highlight some major industrial stakes and challenges related to complex aeronautic systems and related to the digital engineering systems that support their development. This chapter aims at clarifying the focus of this PhD thesis and at deriving, from the identified stakes and challenges, the key technical requirements and issues that our research work addresses.

4.1 Synthesis of key technical requirements

4.1.1 Requirements related to the use of the DMU for CAD-FEA integration

As previously stated, depending on the discipline and the type of performed analysis, digital integration chains and related numerical simulations require defining specific product design models (using specific representation of the product) in order to create the appropriate simulation models. Therefore, using the DMU as the principal source of CAD data inputs for FEA requires a flexible management of the DMU environment. Only a few and recent research works highlight the potential for the DMU for being the backbone of design-simulation loops and to be adapted for domain-specific engineering needs and especially for simulation needs [Drieux, 2006, Nguyen Van et al., 2006b, Foucault et al., 2011, Shahwan et al., 2011a, Boussuge et al., 2012, Shahwan et al., 2012, Shahwan et al., 2013]. Nevertheless, DMU is often used as input for structural analyses of a whole assembly or for thermal and CFD calculations. Below Figure 27 illustrates this high-level requirement by displaying two adapted DMUs from a referential DMU: one for the creation of the mechanical IFEM used for a structural analysis of the IPSS, the other one the CFD calculation of the nacelle ventilation in the core compartment.

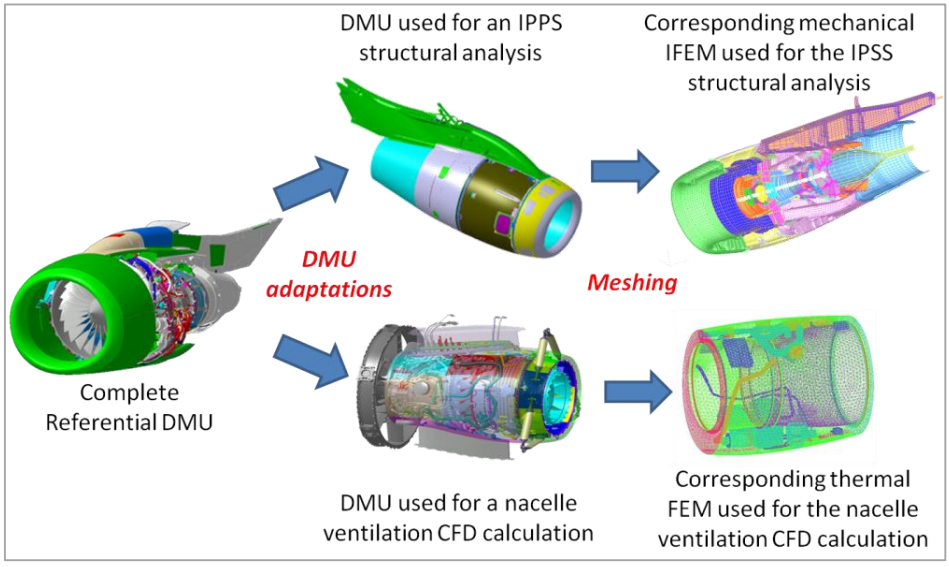


Figure 27: Illustration of required simulation-driven DMU adaptations for two different FEAs

Assumption:

- The DMU is consistent: we are only interested in consistent DMUs that represent a functional product and contain no contradictory information. This assumption allows us to derive reasonable conclusions.

Adapting DMUs represents a very time-consuming and tedious effort due to the technical gaps presented below.

4.1.1.1 DMU shapes transformation

Design models are usually refined for the purposes of manufacturing, and therefore contain fine details that are part of the as-manufactured component. These details can complicate the mesh generation process. Therefore, generating models for FEA from the B-Rep CAD models composing the DMU requires shape transformations to remove details or idealize sub-domains. The principle of removal details, as shown in Figure 28, consists in removing or modifying details in order to simplify the simulation model without affecting the results of the analysis. In the case of the structural analysis, the details to be eliminated are defined by entities of small size which are not carrying boundary conditions (solicitations), not subjected to stress concentrations and which do not influence the deformation and stress field in the remainder of the part. In practice, the analyst will eliminate the details such as holes, embossing and chamfers.

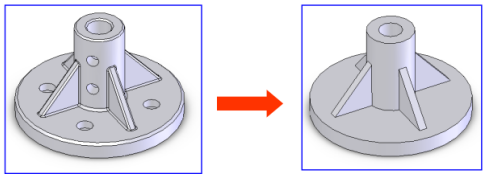


Figure 28: Principle of removal details at component level from [Hamdi et al., 2007]

Hence, the resolution of the geometric model is coarsened so that it more closely matches the objectives and the fidelity level of the target analysis. This simplification process reduces the subsequent mesh generation time, for little cost in terms of analysis accuracy. Currently, these simplification tasks are mostly manual and already very tedious for small assemblies with tens of components and it is not achievable for very large ones because most of the processing is performed interactively by structural engineers. In order to decide whether and how components can be idealized, analysts refer to the type of simulation performed, to the functions of the components and to the level of abstraction of the system/sub-system which is analysed. For instance and as shown in Figure 29, a whole engine model is created from individual engine casings. Those casings contain details of minor importance at whole engine level, although generating a lot of extra nodes, and, as a consequence, degrees of freedom, which has an impact on computation times. Without the removal of those details, the whole engine computations could not be performed.

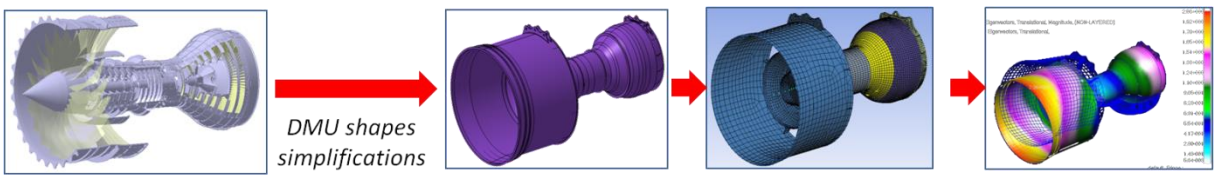


Figure 29: Example of DMU simplifications required for an aero-engine structural analysis

The related technical requirement is to automate these DMU shapes transformation to meet the engineer's requirements in terms of FE mesh generation, simulation objectives, accuracy and time to fit into a Product Development Process (PDP) [Boussuge et al., 2012]. However, this requirement (automating shape simplifications) is out of the scope of this PhD, but the management of the resulting

idealized/simplified DMUs and related CAD models within PDM and/or SDM systems will be addressed in view to be integrated with their related simulation data.

Assumptions:

- The geometric models present in the "referential" DMU are highly detailed ("as-manufactured").
- An increase of computer resources which would make detail suppression useless will not happen in the next few years.
- We cannot rely on the designers (and surcharge their work) to prepare and provide simplified CAD models (already hard inside a company, even harder in a collaborative environment).

4.1.1.2 Simulation-Driven DMU structures

The DMU of a complex system is generally based on a very large product structure, referencing thousands of CAD models. Very few analyses require the whole product DMU. Most of them require a subset of it. For instance, and as shown in Figure 30, the finite element model used for the structural analysis of an aero-engine is created using different modelling dimensions. The structural elements (in blue and beige in Figure 30) have two different types of modelling (3D shell elements for the stator components and 1D/2D elements for the rotors). Other rotor elements like the blades (transparent in Figure 30) can be modelled with a 0D punctual mass element corresponding to the total mass of the blades crown applied on its centre of mass. In that case, only the 3D and 2D elements use 3D or 2D CAD models as input to be created. 1D and 0D elements are created based on the mass or stiffness properties of the corresponding components. Therefore the DMU used for the creation of the FE model does not require containing the components modelled in 1D or 0D. However the analyst needs to easily access the mass properties of these elements.

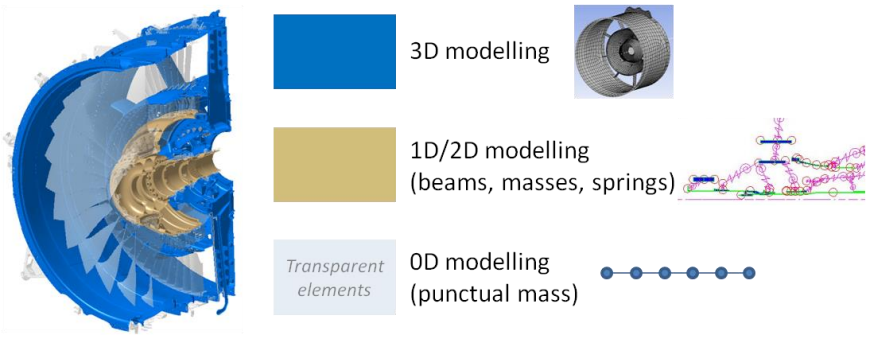


Figure 30: Fan assembly example of different modelling dimensions for the structural analysis of an areo-engine

Preparing this kind of subset and maintaining it up to date quickly turns into a time consuming task. As-is, the DMU tools only provide features allowing to create "on the fly" the subset of the DMU contained in a geometric envelop. As such, either the user prepares a very detailed envelop (time consuming) either he gets useless parts.

Moreover, analysts often need that the structuring of the components represented in the DMU matches with the structure of their simulation model. It is obviously not the case in As-Is DMUs. Rearrangement of DMU product structures is also a tedious, time-consuming but essential activity to prepare the simulation model. For instance, one important input for mechanical simulations is the mass properties of the various components; if the structure of the DMU does not match with the structure of the simulation model, they have to reorganize it so that the mass properties (mass, centre of mass and inertia moments) values of assemblies specifically defined for the analysis are correct. The current DMU structure used for an aero-engine is generally organised into functional modules (FAN

module, Compressor module, and Turbine module) and sub-modules (FAN assembly and inlet, Low pressure compressor /Booster, High-pressure compressor, Low-pressure turbine, High pressure turbine, etc.) where the place of the different modules and sub-modules in the DMU structure can vary in function of the established partnership DMU's management rules. In this kind of organisation stator and rotor components of the engine are spread within these different functional modules. To perform mechanical analyses, the engineer has to reorganise this structure in a rotor-stator breakdown matching with the structure of the integrated FE model. Indeed the FE representation of a rotor and a stator is not the same and the engineer can require for instance to isolate a rotor assembly for measuring its inertia moments. Figure 31 illustrates this issue on a FAN assembly, displaying the modular reference DMU structure on the left side and the structure used for mechanical analyses.

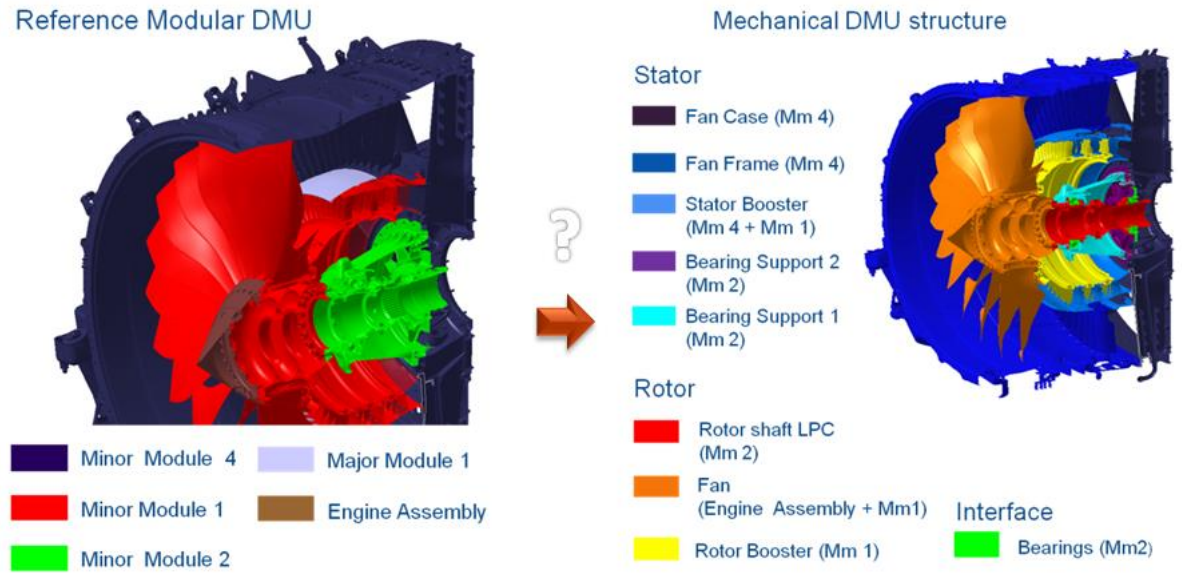


Figure 31: Functional modular reference DMU structure Vs Mechanical DMU structure

Reorganizing a DMUs representing a large system is a very time-consuming and tedious effort due to the following issues that currently impact such usage of the DMU:

- DMU has often inconsistencies regarding the functional description of assemblies resulting from geometric inconsistencies.
- Current PDM configuration management capabilities that permit to structure and make the DMU evolves, only allow the management of customer-oriented product variants. There exists a crucial need to extend the concept of structure nodes/links effectivities for discipline-oriented product variants.
- Current engineering change management process is not often synchronized with the DMU content; there exists a time gap between the moment a CAD design change is proposed and validated and the time the new related CAD model revision is integrated within the DMU. And even when it is synchronized, the change propagation rules and methods across all the potential product DMU representations are not ensured.

As a consequence, both from shape and structural points of views, the representations of an assembly are multiple. Therefore, handling shape transformations of assemblies as well as their structure transformation requires specific operators in addition to lacking of topological description as well as structure transformations, many operations are manual and very time consuming.

4.1.1.3 Explicit Functional and Topological description of Assemblies in DMUs

As above stated, The DMU tends to become a central source of data for the analysts. However, it lacks this crucial information about how the parts interact (not even mentioning how the parts interact with fluids). Interaction is known by the end-user, which has to manually create the links. This effort is reduced by functionalities available in CAE systems consisting in pairing coincident faces. However the system cannot guess what is the mechanical behaviour (glued, sliding contact, interference fit, etc.). This step is manual, time consuming and error prone. The interfaces are also used to ensure that it will be possible to assemble components.

In assembly modules of CAD systems, mating conditions are in fact geometric constraints but have no connections with the topological model of an assembly. Indeed, many FEAs are strongly based on the explicit topological description of an assembly, which means that the topology of links between components should be available [Hamri et al., 2008]. Organization of models in a DMU is defined with respect to the reference frame of the assembly. Hence, there is no geometric constraint between the components and the relative position of components may be subjected to errors. Concerning large systems it is difficult for the integrator to update a set of constraints when there is a modification on components that impacts several geometric constraints. As a result, the interaction areas between components are not captured in most DMUs, while this information is required to represent intrinsically system interfaces. Even though CAD or PDM systems can incorporate assembly modules enabling to define the mating conditions between components, these conditions are limited to the identification of the faces of the components involved in a contact. This information is already interesting but its efficiency is limited in collaborative and distributed design environments because its transfer through standards cannot be performed easily and this type of information is frequently not available in DMUs [Drieux et al., 2007].

The key information needed to transform CAD assembly model into an integrated FE model, are the precise area of the contact area between components as well as the expected behaviour in this area. The contact areas between analyzed component its surrounding ones are of high interest when defining the boundary conditions (BCs) for the analysis of the studied component since they are the locus of interaction forces between this component and its surrounding ones. Therefore, when expressing hypotheses to model, idealize, approximate the BCs, it is critical to precisely define this contact area for modelling a pressure area or considering this area as a boundary between two different types of materials when there is a cantilevered BC between two components or reducing this area to a point if a concentrated force/mass is acceptable to model the interaction between the two components. Figure 32 illustrates the issue representing a CAD bearing sub-system model and its equivalent representation for a specific structural FE analysis.

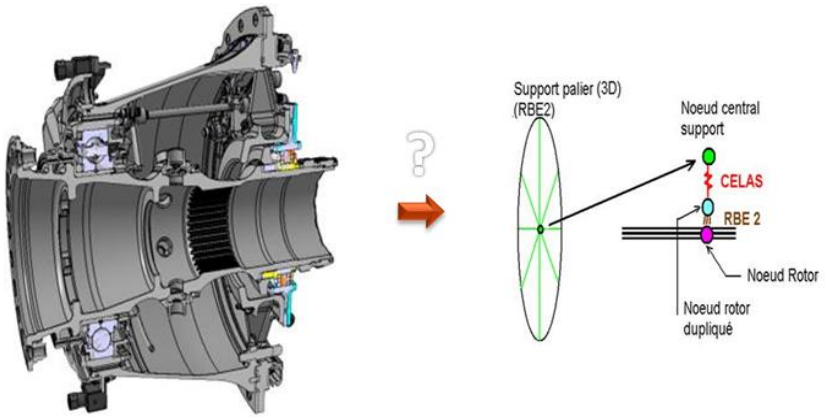


Figure 32: CAD Bearing sub-system and its corresponding FE 1D representation for a specific structural analysis

Without more topological information besides geometry, model positioning and the asscoiative links between CAD models and FE model, it is a tedious work for the analyst to identify within the DMU the contact areas where he has to apply BCs and what are the coordinates of the rotor node he has to project on the engine axis.

Considering these technical issues, the ability of a DMU to model a system from multiple viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail, fidelity and abstraction is of importance today. However causes of these issues have emphasised necessary improvements in the management of complex systems architectures modelling techniques and on current PDM information models and capabilities.

4.1.2 Requirements related to system architecture modelling capabilities with an object-oriented approach

Most of system architecture modelling approaches concern functional, structural and behavioural architectures and intend to describe the relationships between functions, behaviours and structures during the design process. As previously mentioned, the modularisation of product architecture (intending to reduce the number of interfaces between modules and to handle more easily the structural complexity) introduces challenges in integration phase because of the difficulty of predicting possible integration impacts due to interfaces between sub-systems.. Integrators face many difficulties to retrieve and gather the appropriate design information and to manage the sub-systems interactions in order to assemble integrated product. Most of rework is done due to inadequate input data to perform integration and simulations. **Indeed, during system integration, issues often arise from unsuitable components models, from interactions that are often insufficiently (or improperly specified) and/or from unsuitable structure and view of the product.** That is why it is desirable to provide to integrators and analysts a view of the system that is tailored to their particular task or design context.

A key issue in dealing with multiple views is that different views of a system all relate to the same system and thus depend on each other. However, since different information is represented across multiple views, only portions of different views are related to each other [Shah et al., 2009]. Consequently, a mechanism is needed to integrate the required information between views. In current practice, the views are usually represented in different tools and are often maintained independently of each other by the users, resulting in significant non-value-added effort and in significant opportunity for errors. In addition to integration between domain-specific views, a mechanism for multi-level system design is required to ensure the continuity of information between views at different system breakdown levels. This integration is difficult to handle due to a lack of systems-level modelling capabilities. To overcome all these problems, three different approaches are commonly discussed in the literature:

- The use of customized mappings between different product views and structures;
- The use of system modelling language (e.g. SysML) and formalisms for both multi-level system design and for integrating multiple domain-specific views of the system. These languages and formalisms aim at providing the designer the ability to maintain bidirectional consistency between models from different domains [Shah et al., 2009].
- Models interoperability approaches: consisting in exporting the system model into standard file formats to achieve single-direction integration.

These approaches aim at enhancing information exchange between domains and to allow designers to start from a systems perspective and automatically generate domain specific models that are necessary for the latter stages of the design process. The difficulties encountered to implement such

an approach are due to a crucial lack of integration between definitions of design contexts, configurations and processes' activities with the existing product data and their relevance in those contexts. Currently these product data are neither managed consistently regarding the design context for which they are relevant (except customer-oriented product configurations managed through effectivities), neither controlled through an integrated reference framework that permit to display the appropriate view of the product to use in the specified context.

Considering the use of such system modelling approaches for simulations and especially in FEA, the functional, behavioural and structural aspects of a complex system can be viewed and understood differently regarding the discipline of the simulation. Therefore the simulation model must represent the product view that match with this discipline, by gathering relevant design information of the product and the studied breakdown structure. Current EDM systems require intelligent objects and operations enabling to provide the suitable and reliable CAD and DMU data inputs regarding the specification and the purpose of simulation (mismatch between study requirements and data used to perform the simulation). They also have to support the interface specification that includes multiple levels of hierarchy and multiple level of abstraction in order to optimize the integration/assembly activities whether it is for contextual CAD design of assemblies for the creation of Integrated FEM (IFEM).

4.1.3 Requirements related to Multi-Aspect information structure in existing Product Data Models and PDM/PLM systems

4.1.3.1 Multi-view and Multi-domain Product data models

For a complex product where multidisciplinary engineering teams are involved, product data models must support collaborative design, hence the link between the various representations of the product handled by designers. This model has to be able to cover all experts' intents. Indeed, each expert studies the product for a particular perspective (or with different objectives) and gives his special description of the product [Labrousse et al., 2004]. Indeed each designer represents a product with different elements and different hierarchical compositions. Each representation is context dependent. It is thus a unique view of a product configuration. This context dependence comes from the fact that design objectives are different and therefore both modelling rules as well as design parameters taken into account will have a certain angle. Each point of view and each representation must be integrated into a coherent representation of the product.

With product development evolution, product data will change in numbers and types. Therefore, there is a crucial need to be able to encapsulate the tools, data, models, methods, and other resource objects that are used in a single specific engineering discipline and/or a specific lifecycle stage. There exist different product data models used for different engineering domains. Product feature attributes describe product data objects. For an engineering domain, product feature attributes describe the product data objects for this domain. From the perspective of object-oriented technology, product feature attributes are integrated by the data subset of product data models in every domain. So both mapping of product data structure between engineering domains and mapping of product featuring attributes are required [Li et al., 2011].

Unfortunately, product data model currently implemented in PDM and PLM applications do not take into account these multi-domain and multi-view notions, although these notions are fundamental to efficient integrated collaborative design. The challenge still remains as to how to connect the representations of product models and actual products in a structured and formal fashion so as to accomplish the harmonization. This harmonization enables to define links, allow the exchange of information

and ensure consistency between views and domains in integrated environments. These links can also support the process of propagating engineering changes across these multiple product views.

4.1.3.2 Engineering change propagation

Engineering changes (ECs) provide a unique and official channel in a company for change propagation between multi-level and multi-domain design teams and from design to other departments (e.g. manufacturing or customer support departments). However, ECs can also be a major source of inconsistency if they are not properly propagated to the collaborating departments. Therefore, companies require product data views for each department that support their specific views and EC propagation procedures that maintain consistent product data between design departments [Do et al., 2008]. Therefore, for consistent EC management between multi-domain or multi-disciplinary design teams, ECs should provide product structure-oriented representations, and "effectivity" management for domain-specific product views, integrated objects for workflow applications and integration with product configurations. Existing academic research efforts have paid little attention to the required integration between the definition of domain-specific engineering contexts and related product data views to the EC process and more especially the change propagation channel for data consistency maintenance between multi-level and multi-disciplinary design teams. These industrial requirements are addressed by international standards such as ISO 10303. However, current industrial implemented methods and used commercial EDM applications are far from fulfilling these requirements.

The EC propagation issue through the multi-level and multi-domain product representations and related structures is closely related to the topic of interfaces data management and CAD-CAE data associativity. Indeed, information exchange and design changes impacts effects between multi-level and multi-domain design teams generally occurs at what we have called product interfaces. Moreover when a design change occurs, there is a critical inertia phenomenon between the moment the change is proposed and the moment the change and its related impacts are validated by simulations. This is due to the fact that EDM applications and product data models do not ensure explicit links between the objects defining the EC intent (that can be for instance linked to another EC object, or to answer new engineering requirements), the CAD data features and parameters that physically characterize it and the current domain-specific simulation models and results that could be impacted by this EC. Interfaces objects and functions can also be used to predict and support change propagation since a change on the design of one component or sub-system might have impacts on the design of the neighbouring components or sub-systems because of the potential modification of the features of the interactions that take place at these interfaces.

That is why before considering the use of digital engineering change objects according to an extended usage of product configurations and "effectivity" management methods in EDM applications, two requirements must be addressed:

- The use of digital interfaces objects as corner stone objects in product data models in order to efficiently capture product assembly and interfaces information in PDM and DMU environments;
- The associativity between design and analysis data: it means the possibility of linking them through a product data or product meta-data aspect (e.g. linking a CAE model with the CAD model used to generate it), or through a parametric or topologic aspect (e.g. linking a geometric face of the CAD model to the corresponding elements in the FE models generated from it).

4.1.3.3 Capturing product assembly and interfaces information in PDM systems

The issue of exchanging parts and assembly information between modelling systems is critical for unrestricted exchange of product data. An assembly information model contains information regarding parts and their assembly relationships (hierarchical relationships, assembly features, kinematics joints, topology of interfaces, etc.).

Interface management is a cornerstone of systems engineering. Stevens argues that interfaces must be clear, kept stabile, and be kept separated [Stevens, 1998]. A common method to manage interface information is to specify the intended interactions in an Interface Control Document (ICD). This is a coarse-granular and document-centred approach, which offer limited possibilities to efficiently support or automate down-stream life-cycle activities.

A more fine-granular approach is to treat each interface as an object that relates a pair of mating features, located on different material objects, i.e. features on physical parts, human organs, or the environment [Sellgren, 2009]. The challenge is to define and capture interfaces features (defines in CAD and CAE models) that can be stored and managed as fine-granular explicit models by a centralized product data management (PDM) system. Present mechanical CAD-technology is generally feature-based, which makes most CAD-tools suitable for representing geometric mating features and in some cases also non-geometric properties can be represented and managed. A severe limitation in present PDM/PLM-systems is that part features are not represented in the standard information models. This limitation is a major obstacle for managing fine-granular systems models that are composed of component models (material objects or parts and mating features) and interface models [Sellgren, 1999]. These interface data models must be implemented in PDM/PLM systems, but require mating feature and interface extensions, to the PDM information model.

Although product data models were developed in the literature to propose standard representations that specify assembly information/knowledge, these standards are sources of data loss and of information duplication since they are not yet fully implemented in commercial applications. However there is still a lack of capabilities and implementation methodologies permitting to make these models interoperate with current commercial CAD and CAE applications.

4.1.3.4 The various level of Design-Analysis data associativity

Another important capability that product data models must support is the consistent integration of design information from the product definition (e.g. CAD data present in DMUs, Bill of materials) with the associated simulation data in order to ensure a complete traceability of the design/simulation information chain and facilitate the management of change impact in PDP. This capability can be managed through high-level traceability between CAD-CAE metadata objects.

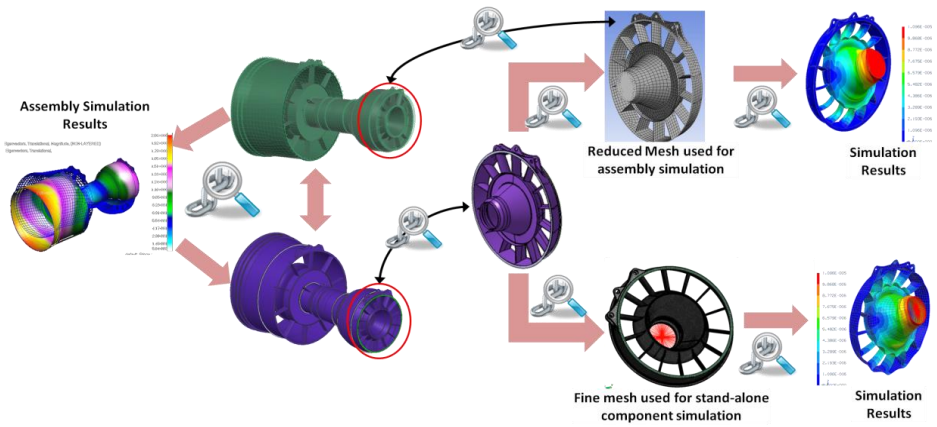


Figure 33: High-level traceability between CAD-CAE metadata objects

Figure 33 highlights the missing traceable links that permit to retrieve which CAD data has been used as input for a specific CAE activity. These traceable links can enhance re-use of models (avoiding re-work) when information about the context in which the data have been used is provided. One CAD model can be idealized differently to create different meshes for various levels of abstraction. Several meshes can be generated from one CAD model depending on the needs of the users (disciplines, level of detail…). The goal is that any mesh created is managed and related to the geometry it is representing. It is necessary to easily navigate through this CAD-CAE models network in order to find quickly all existing meshes for a given CAD model. This will avoid mesh creation redundancy by increasing the awareness of the user about existing meshes, and more generally decrease mesh generation effort (avoid re-work, ease re-use) and increase modelling quality. Moreover, in CAD parametric modelling, another tricky challenge is to manage the association between specific analyses results and geometric parameters and features to make them evolve according to the results of the simulations.

In modern CAE environments, the user interacts indirectly with the mesh through the geometry. For example, boundary conditions are applied on faces of the geometry. It allows also being able to define uniquely the boundary conditions applicable to all the different meshes associated to the geometry. Therefore fine associations between the FE mesh and the topology of the CAD model it is representing (vertex, edge, face) are also required. Figure 34 gives a concrete example of that kind of issue. Depending on the simulation intent [Nolan et al., 2011] and the complexity of the geometry, different types of geometric areas can be defined on a CAD model and these areas can be meshed differently. If for instance a face on the 3D model need to be transformed into a mid-surface to be meshed, the association between the face on the 3D model and the equivalent edge on the mid-surface is required.

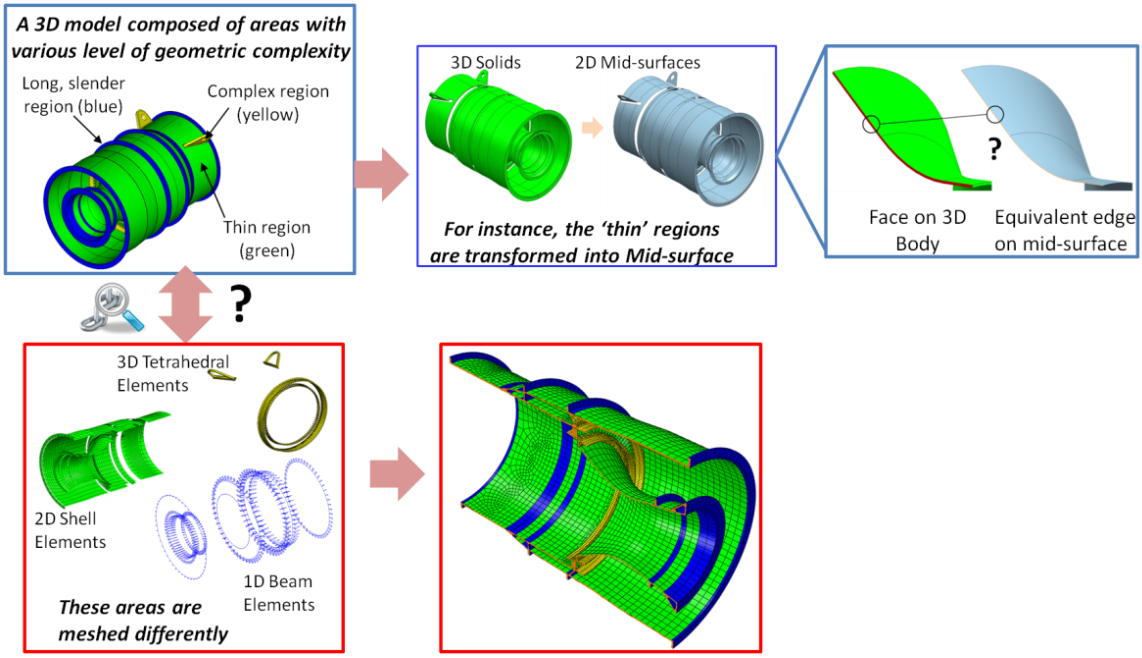


Figure 34: Example of required fine grain CAD-CAE association adapted from [Nolan et al., 2011]

If product data models cannot ensure these levels of traceability between CAD and FEA data the links between states progress of the design model and calculation remain uncertain. The lack of traceability also complicates the reconciliation between product changes and their impact on the analysis results. This has resulted to affect the confidence in the analyses results. In addition, it is important to provide rapid access to critical data for the two activities to prevent from going too far in a wrong direction.

4.2 Conclusion and research objectives

Previously analysed industrial requirements have been highlighted in order to propose further development (framework) to enhance digital integration chains and design-simulation loops efficiency. Figure 35 illustrates the synthesis of previous section regarding the industrial requirements and highlights the need for a multi-view, enriched and flexible DMU environment supported by intelligent multi-aspect product data models and system modelling frameworks.

In terms of research objectives, and to fulfil these requirements, this PhD intends to answer the following research questions:

- How to ensure a more rapid and easier acquisition of simulation data inputs making the DMU more flexible so as to be used as the main model-based source of information?
- How to identify the relevant data set to be used for the simulation and to organize this data set into a new adapted product structure and "engineering environment"?
- How to ensure a more efficient and consistent integration of adjacent product components models so that integrators and analysts can numerically and easily verify the behaviour of the integrated product and reiterate faster within design integration chains?
- Furthermore, integrating different product components in complex system design is iterative and often produces large scale intermediate data with heterogeneous formats and complex relationships. The efficient organization and management of engineering data are therefore a bottleneck of product design performance and this topic is discussed in this PhD around two issues:
  - How to ensure the continuity of information between working teams and more especially between design and multi-disciplinary analysis models/data?
  - How to enable PDM systems and surrounding CAX applications to display appropriate system representations from multiple viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail, fidelity and abstraction?

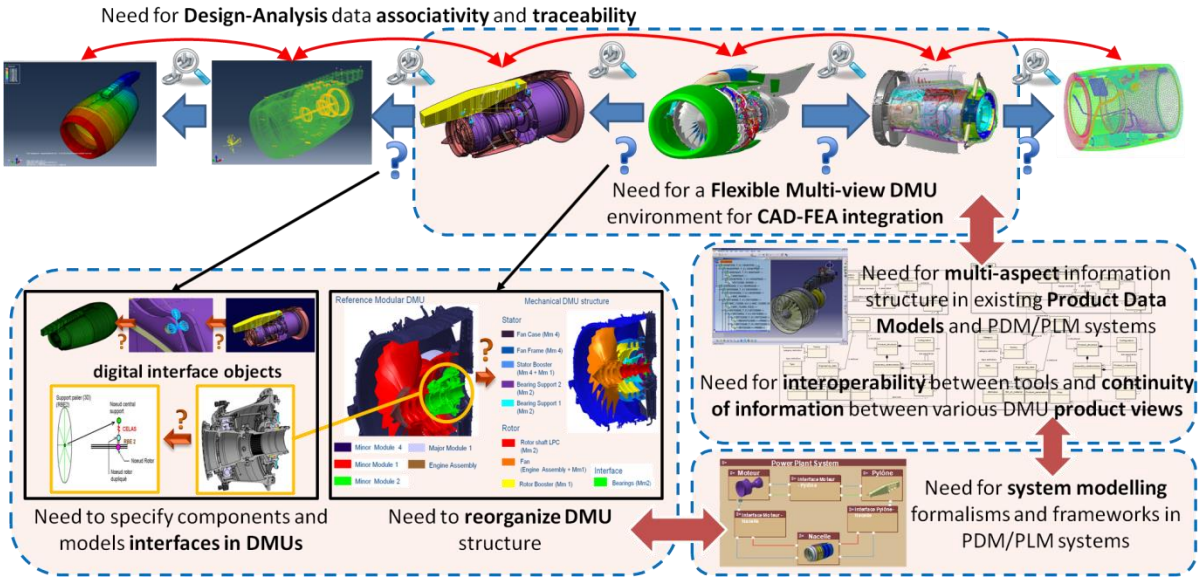


Figure 35: Synthesis of industrial requirements to optimize digital integration chains

PART II: STATE OF THE ART

In order to achieve the research objectives and issues mentioned in the previous part, it is important to position our research works regarding the literature and to identify the related works that have already addressed such issues. This part corresponds to the state of the art of this PhD and is made of 4 chapters.

The first one (chapter 5) introduces our scientific positioning and provides a clear understanding of the structure and content of the following state of the art.

The following chapters address respectively the main research areas mentioned in Chapter 5 underlining the most interesting works as well as the current related gaps and limitations.

Chapter 6 addresses the topic of Model-Based Systems Engineering (MBSE) since it appears as being the most appropriate approach to describe a system from different viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail. It hence provides an overview of ongoing research works and MBSE concepts, methodologies and formalisms that intend to provide a consistent, interoperable, and evolving model of a system throughout its lifecycle.

Chapter 7 specifically deals with the topic of using the DMU as a main product definition referential for implementing a MBSE approach in simulation-based mechanical (in opposition to software products) product design. It provides a literature review of existing approaches, methods and tools aiming at ensuring the continuity and traceability of information between CAD models, their assemblies in DMUs and the simulation data and activities that use them as input. We are particularly interested in existing works dealing with the issue of adapting the content and structure of DMUs according to their usage in the product lifecycle.

Since a DMU is supported by a PDM a system, Chapter 8 provides an exhaustive review of existing product data models enabling to manage and represent product design information according to various discipline-specific aspects of a design artefact. This includes generic product data management capabilities, explicit description of assemblies' structures and components' interfaces, the continuity and traceability of CAD and CAE data at different levels of abstraction and the re-use of design artefacts in the appropriate context. The exchange and consistency of information as well as change propagation across multiple multi-domain and multi-level views of the system are also addressed.

Chapter 5: Scientific positioning

The aim of this chapter is to justify and clarify the scientific positioning of this PhD dissertation and to provide a clear understanding of the structure and content of the following state of the art. Adopting a lean approach, we have investigated the value chains that operate in system engineering and integration processes of aeronautics products such as aero-engines. The results of this analysis are summarized in the first section of this chapter by first introducing the Lean Product Development principles, defining what mean value and waste in PDP and finally identifying the main related value and waste drivers. As explained in section 5.1, this analysis has permitted to justify the importance of addressing current design-analysis integration issues for achieving the objectives of lean engineering. That is why section 5.2 introduces the various design-analysis integration research areas to investigate and provides the structure of the following state of the art.

5.1 Lean Product Development and Simulation-Based Design

5.1.1 Introduction to Lean Product Development

Since the 90's and the publication of "the machine that changed the world" [Womack et al., 1990], the lean approach has proven its positive results concerning efficiency and reduction of overall business process time. Product Development Processes (PDP) has an important role in the value definition since it aims at defining the product and customer value, and this definition largely impacts production costs and production times. Therefore, in the past few years industrialist and researchers have shown a great interest in transferring the lean principles to PDP; called Lean Product Development (LPD) or Lean engineering.

Lean Product Development (LPD) consists in creating the right products by first managing effectively the product lifecycle and the enterprise integration and by using efficient engineering processes and applying lean thinking to eliminate wastes and improve cycle time and quality in engineering [Mc Manus et al., 2007]. **Lean thinking** is essentially a corporate culture oriented towards customer satisfaction in terms of added value as well as in terms of permanent reduction of the time required for creating this added value. Therefore Lean philosophy consists in doing the "just needed" to create the desired value. This "just needed" can only be achieved through the identification, monitoring, analysis and continuous improvement of value chains in the company. As shown in Figure 36, the purpose of this approach is to create a continuous flow of material and information to deliver the desired customer value with the least possible waste of resources and minimized delays.

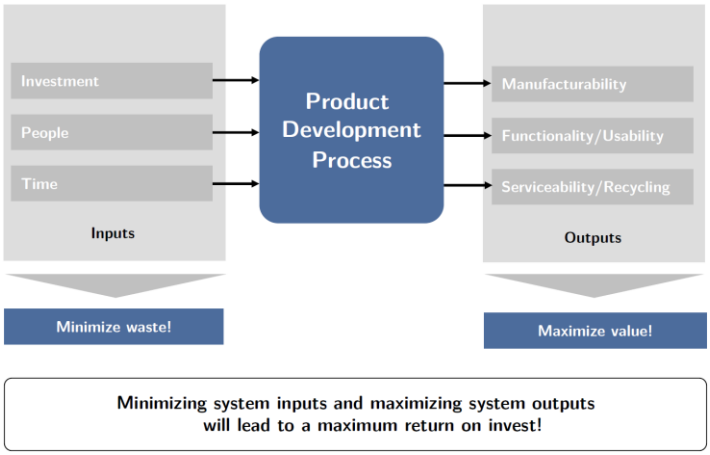


Figure 36 : The Lean Product Development scope (extracted from [Hoppmann, 2009])

In "Lean Thinking", [Womack&Jones, 2003] define an iterative approach, inspired by the Deming's "quality wheel", sequenced in 5 steps called the 5 basic lean principles:

- **Defining what creates value for customers**: customers must not pay the additional costs of products and /or services purchased to compensate for its supplier inefficiency and waste (with the risk to see the client going to a more competitive supplier). To set the value correctly will also permit to see if we make over-quality.
- **Identifying the value stream**: it means mapping processes and identifying added value activities and non added value activities that waste resources.
- **Promote the flow of the stream by ensuring that the stages of value creation are optimized**: once the sequencing of the tasks is set for an optimized workflow, it is necessary to standardize and make processes transparent in order to be able to monitor and control them.
- **Pull the flow downstream**: the production is triggered only after a customer order (with some safety stock to overcome all unpredictable variations in demand).
- **Striving for perfection to achieve excellence**: repeat the loop indefinitely in a continuous improvement process.

Value definition is the starting point of the lean approach. However, contrary to production, PDP are multidirectional, processes and process chains are highly connected, and the feedback loops and iterations intersect at multiple hierarchical levels. [Chase, 2001], [Bauch, 2004] and others consider PDP as an "information creation factory", hence, "the product of product development is information". Assuming that, lean engineering is not based on a product and customer oriented value definition but more on a knowledge/information oriented value definition. **The PDP value chain hence operates through the different information flows containing the engineering product data and enabling product and process knowledge capture.** Before applying the other lean principles to PDP, it is therefore crucial to define what value and waste mean in product development.

5.1.2 What and where are the value and waste drivers in PDP?

In [Womack&Jones, 2003] authors state that only the customer can define the value, which is "a capability provided to a customer at the right time at an appropriate price, as defined in each case by the customer". This definition, according to Chase, is useful for applications where the final product is explicitly defined, such in manufacturing [Chase, 2001]. Therefore, in view to the optimization of PDP, he proposes a more specific value definition considering product, process and organization dimensions and integrating different perspectives. The same author proposes a framework that underlines that **PDP activities aim at increasing information and knowledge about the product definition in time while reducing risk and uncertainties on the product, hence on the project** (see Appendix III, Figure 243)

The question of waste definition is crucial if one wants to contribute to the global optimization of the PDP system and to increase this process efficiency. What is the waste in PDP? And what are the related waste drivers? The first question was initially addressed by transposing seven types of production waste to the area of PDP and eventually adding some categories [Womack&Jones, 2003], [McManus&Millard, 2004]. Morgan considers other specific types of waste, but he has sometimes mixed types of waste and waste drivers [Morgan, 2002]. In [Bauch, 2004] and afterwards in [Kato, 2005], authors map all the ideas from the previous studies in order to identify all potential type of waste. Bauch has elaborated a cause and effect diagram in order to distinguish waste types from waste drivers. This diagram is given in Figure 245 of Appendix III. In [Wenzel&Bauch, 1996], authors highlight one of **the most significant problems and waste driver within PDP: uncertainty and risk**. They argue that, at any point of time in the PD life cycle, even if the characteristics of a system are planned and

specified, the actual knowledge about the system at this point in time, gained and confirmed through testing and verification, always falls below the planned level. Consequently, there always exists a **knowledge gap** between assumed and verified characteristics (see Figure 37). Since it always lasts a while until planned features are verified, there also exists a **time gap** between assumption and verification. During this time, design activities may be based on wrong assumptions what in turn can end up in loops and high levels of rework that can be considered as waste if this rework could have been avoided. **To conclude, uncertainty and risk are the main waste drivers in PDP and everything that generate uncertainty and risk can be considered as a potential waste driver, whereas everything that permit to reduce this uncertainty and risk gap is considered as a value driver.**

In Simulation-Based Design (SBD) (see section 2.3.2), **the knowledge and time gaps above mentioned can mainly be observed and addressed within iterations between design and simulation activities.** In chapter 2, we have already underlined the importance of these simulations for efficient system integration. Section 2.3.2 particularly underlines the reasons for adopting a SBD process in such a context: “when coupled with appropriate validation processes, the resulting capabilities can provide companies with the ability to design superior products in less time and at lower costs” [Shephard et al., 2004]. **Therefore, we argue that simulation-based design is an inherent part of lean engineering applied to the design and integration of complex systems**. Hoppmann, inspired by the "*The Toyota Product Development System*" and its related 13 components defined by [Morgan&Liker, 2006], corroborates this argument by defining “Rapid Prototyping, Simulation and Testing” as one of the eleven LPD system building blocks [Hoppmann, 2009].

As already mentioned, the numerical simulations performed in a SBD process aim at verifying that the product definition satisfies the expected requirements; that the anticipated/expected product knowledge correspond to the verified knowledge: the product behaviour. The related design-analysis iterations (or design-simulation loops) and the way they are handled are therefore of primary importance. In Figure 37 below, we have drawn a parallel between a simplified and traditional PDP process (where design iterations are highlighted in red) and the graph representation of the knowledge and time gap from [Wenzel&Bauch, 1996].

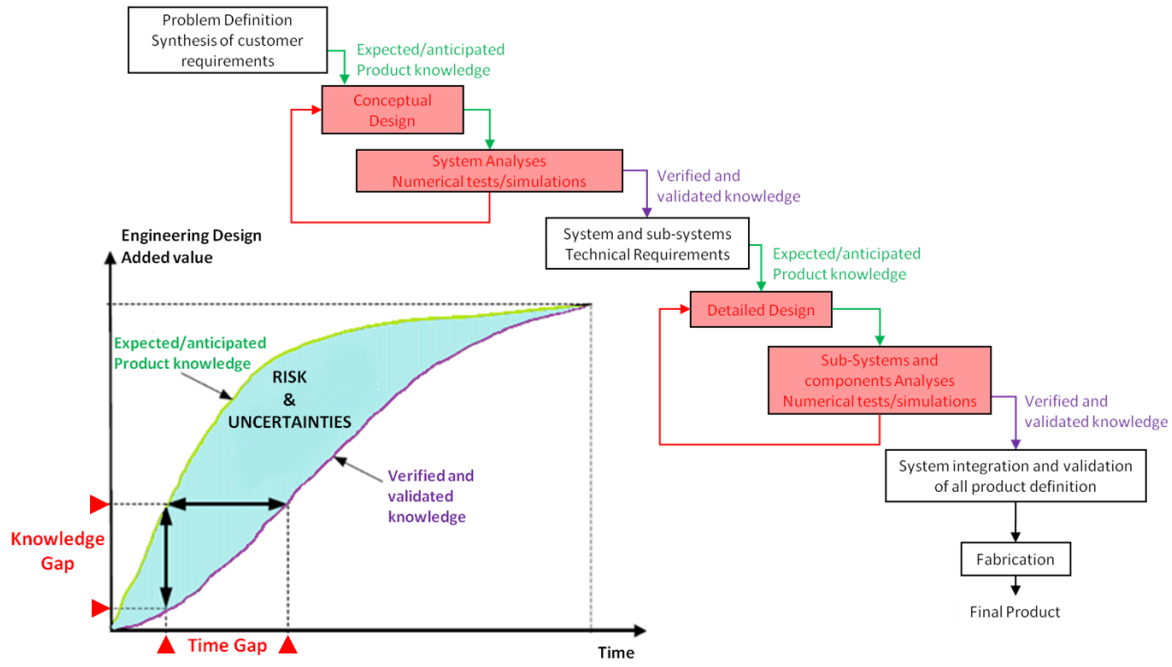


Figure 37: Time and knowledge gaps to reduce risk and achieve PDP performance (adapted from [Wenzel&Bauch, 1996])

Figure 37 illustrates one important challenge in PDP: in order to reduce the development cycle time and delivering the expected customer requirements, **the challenge of lean engineering is therefore to reduce this knowledge and time gap the fastest possible to reach the expected product definition with the expected value added**. In section 2.3.2, an example and illustration (see Figure 10) of such knowledge and time gaps occurring between inter-dependant simulations loops of an aero-engine development process are provided. Based on these considerations and based on the synthesis of the industrial requirements described in chapter 4, we believe that addressing current design-analysis integrations issues in PDP is the most important challenge for achieving the objectives of lean engineering applied complex system design and integration. The following state of the art hence focuses on design-analysis integration issues.

5.2 Bridging the gap between Design and Analysis for efficient System Integration

Industrial context analysis shown in previous part has underlined some of the issues related to digital integration chains and more especially design-analysis integration issues. According to a survey by Mocko and Fenves, current design-analysis integrations issues in PDP are discussed in three domains [Mocko&Fenves, 2003]:

- The **Object-Oriented Modelling:** it concerns all methods and techniques for modelling physical systems, with benefits of reusability and modularity that provides an intuitive way to model physical objects.
- **CAD-FEA Integration**: most issues of design-analysis integration are presented specifically related to the integration of CAD activities and models with FEA, since CAD data serve as input for the pre-processing process.
- **Multi-Aspect information structure** researches concern implementations of product data models enabling to manage and represent product design information according to various discipline-specific aspects of the design artefact. These models might also supports evolution of these information and domain-specific views of the product during the entire lifecycle of the product. Therefore, this area is strongly related to product data standards developments and implementations in PDM/PLM environments.

In view to research questions, contributions in this work can be identified and positioned using a framework for Design-Analysis Integration research scope defined by [Mocko&Fenves, 2003] (see Figure 38. We have detailed the sub-topics that correspond to our focus of attention according to the identified industrial requirements introduced in Chapter 4.

In the area of “Object-Oriented Modelling” the following topics are investigated:

- **Model-based system architecture modelling languages and frameworks**: this section includes an introduction to Model-Based System Engineering (MBSE) and object-oriented modelling methodologies. It hence surveys existing graphical modelling languages and frameworks for supporting the practice of MBSE and especially for representing and specifying functional, physical and behavioural system architectures.
- **Continuity of design and behavioural models**: this section includes existing object approaches permitting to link design and behavioural models parameters in order to automate the composition and the update of the models. It also addresses object-oriented approaches permitting to enrich interfaces and interaction information to be able to change components definition without modifying the interfaces definition.

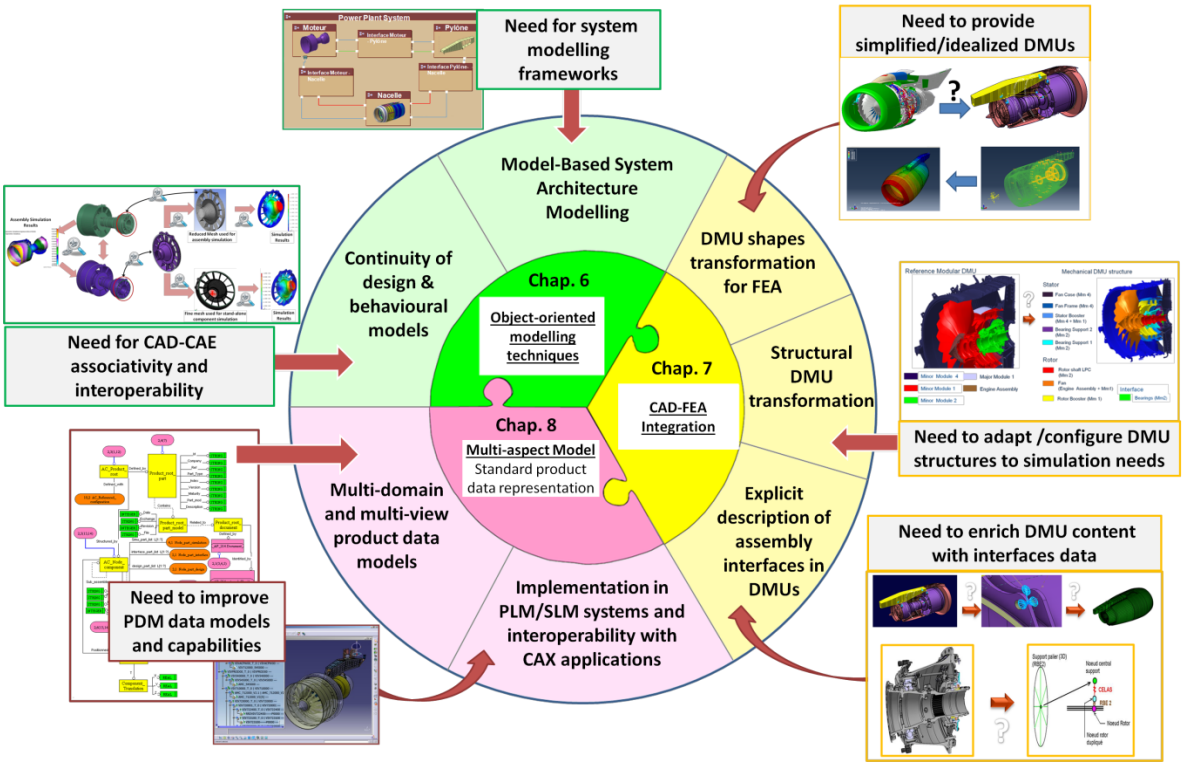


Figure 38: PhD scientific positioning and contributions regarding Design-Analysis integration research areas (adapted from [Mocko&Fenves, 2003])

In the area of "CAD-FEA integration" a special emphasis is done about all the DMU transformations required to provide multi-level and multi-domain DMU product-views in order to make the DMU the central product definition referential supporting model-based and simulation-based mechanical system design. This section hence encompasses the following sub-topics:

- **DMU shapes transformation**: addressing existing approaches and methodologies enabling to automate the details removal, shape simplification and idealization of geometric CAD models that are required for the preparation of usable FE assembly models.
- **Explicit Functional and Topological description of Assemblies in DMUs**: addressing existing approaches that permit to enrich the DMU with the design intent of the assembly which is expressed more precisely by the functional and topological description of interfaces and interactions. This information is missing and required to express the right modelling assumptions and requirements for the creation of FE assembly models.
- **Simulation-Driven DMU structures:** addressing the existing methodologies that support different organizations of the DMU to offer to different actors a DMU structuring corresponding to their point of view and business needs (or simulation objectives in our case).

In the area of "Multi-Aspect information structure" we investigate all existing standardized or non-standardized product data models aiming to be implemented for PDM applications so that they can support the concept of multiple DMU product-views. We believe that this concept, coupled with the use of object-oriented system modelling frameworks, is the missing bi-directional interfacing concept enabling to manage usable Behavioural Mock-Ups and the links with their associated contextual, functional and physical product definition data or meta-data. This chapter hence provides an exhaustive literature review of product data models supporting

- **Generic product data management capabilities**: this includes especially design artefacts definition, components' physical properties and shapes, product structure and configuration management capabilities;
- **Explicit description of assemblies** using specific objects and methods to capture the functional and topological description of components interfaces;
- **The continuity and traceability of CAD and FEA data** at different levels of abstraction and the re-use of design artefacts, models and knowledge in the appropriate context;
- **The exchange and consistency of information across multiple multi-domain and multi-level views of the system** as well as the propagation of engineering changes across these multiple product views.

Chapter 6: Model-Based System Engineering and Integration methodologies

The first part of this PhD dissertation has highlighted that the multi-disciplinary nature of complex system engineering projects results in large quantities of design data, managed in different tools corresponding to each domain. Maintaining consistency between these multiple data sets and tool-specific models becomes an issue when analyzing different system architectures during the design process. The first objective of this chapter is to explain why Model-Based Systems Engineering (MBSE) appears as being the most appropriate approach to describe a system from different viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail. The chapter provides an overview of ongoing research works and MBSE concepts, methodologies and formalisms that intend to provide a consistent, interoperable, and evolving model of a system throughout its lifecycle. Finally current limitations and gaps related to these MBSE challenges are underlined.

6.1 Model-Based Product/System Design Definition

6.1.1 Model-based definitions

Model-Based Engineering (MBE) approach is using models and modelling activities as the cornerstones of the design process [Estefan, 2007]. They are used for the specification, design, integration, validation, and operation of a system; beginning in the conceptual design phase and continuing throughout development and later life cycle phases [ODASD, 2011] [Bergenthal, 2011]. MBE leverages modelling and simulation techniques to deal with the increasing complexity of systems. Models can assist with all aspects of the complex system life cycle, from the interaction of stakeholders in an easy to use environment, to enabling the automatic interaction of sub-modules at different physical scales and across multiple domains [Hamilton, 2010]. In the field of engineering design, MBE is an approach in which models:

- Evolve throughout the development life cycle,
- Are integrated across all program disciplines,
- Can be shared and/or reused across various domain-specific design processes to improve processes efficiency achieve faster expected product quality requirements,
- Can scale up to complex systems and enable to analyze complex relationships and dependencies between design and analysis data,
- Enable visualisation of design artefacts and can be powerful communication enabler,
- Promote automation in modelling and simulation activities

Model-based definition (MBD) is a new strategy of product lifecycle management (PLM) based on CAD models transition from simple gatherers of geometric data to comprehensive sources of information for the overall product lifecycle. With MBD, most of the data related to a product are structured within native CAD models, instead of being scattered in different forms through the PLM database. MBD aims are suppression of redundant documents and drawings, better data consistency, better product/process virtualization, and better support for all computer-aided technologies tasks under engineering and manufacturing disciplines [Alemanni et al., 2011].

6.1.2 Model-Based System Engineering and Integration

MBSE "is the formalized application of modelling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases" [Friedenthal et al., 2007]. MBSE encompass a set of processes, methods, and tools used to support the discipline of systems engineering with a "model-based" or "model-driven" approach. In MBSE, models are used to represent Functional, Structural, Operational and Behavioural characteristics of the system being developed.

MBSE enhances the ability to capture, analyze, share, and manage the information associated with the complete specification of a product, resulting in the following benefits [Murray, 2012]:

- Improved communications among the development teams
- Increased ability to manage system complexity by enabling a system model to be viewed from multiple perspectives, and to analyze the impact of changes.
- Improved product quality by providing an unambiguous and precise model of the system that can be evaluated for consistency, correctness, and completeness.
- Enhanced knowledge capture and reuse through information gathering in more standardized ways and leveraging built in abstraction mechanisms inherent in model driven approaches. This in-turn can result in reduced cycle time and lower maintenance costs to modify the design.
- Improved ability to teach and learn systems engineering fundamentals by providing a clear and unambiguous s representation of the concepts

However MBSE is an emerging practice and little evidence exists that quantifies the benefits. In [Murray, 2012], the author provides the most complete and recent overview of MBSE methodologies recognized by the INCOSE (International Council on Systems Engineering). The description of these methodologies won't be detailed here. However an overview of the system modelling frameworks that support some of these methodologies is given in section 6.2.2. According to these expected benefits and regarding the required high-level capabilities to achieve them, the INCOSE has proposed a roadmap for MBSE concepts and methodologies development. This roadmap is given in the following Figure 39.

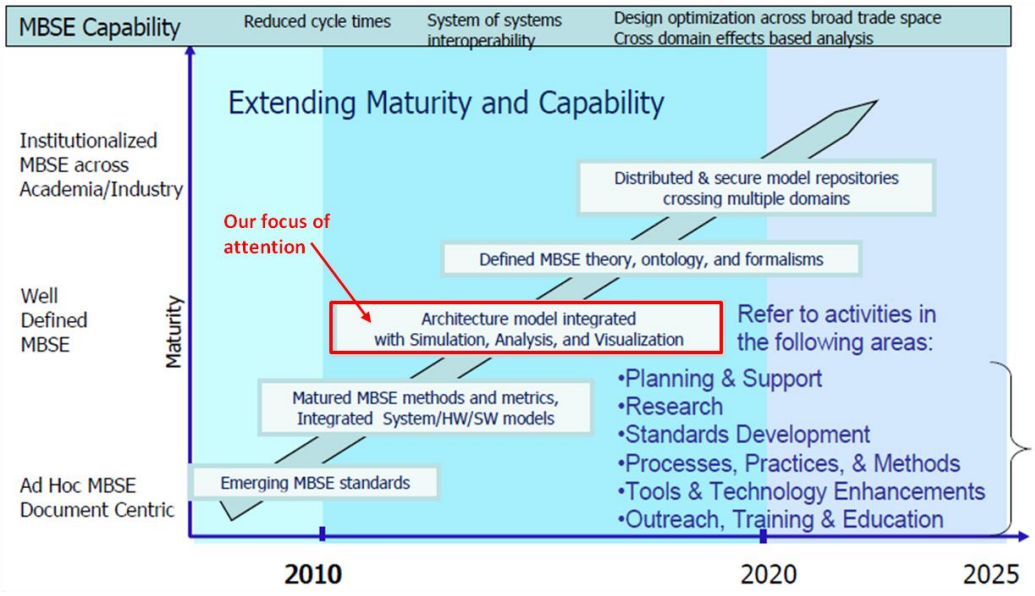


Figure 39: INCOSE MBSE Roadmap from [Murray, 2012]

This roadmap includes the following technical challenges:

- Providing system architecture modelling languages and frameworks to produce and control a **coherent system model**, and using this model to specify/design the system and ensure cross-domain models integration
- Defining domain-specific modelling languages and visualization enabling the systems engineer to focus on modelling of the user domain
- Developing standards based on a firm mathematical foundation that support high fidelity simulation and real-world representations
- Extensive reusing of model libraries, taxonomies and design patterns
- Specifying standards that support integration and management across a distributed model repository
- Ensuring highly reliable and secure data exchange via published interfaces.

The current MBSE methods do not adequately support management of highly complex cross-domain models, including configuration, version, and variant management and reuse of models and the modelling environment, or the ability to propagate changes from one model to changes in other models. Specific gaps to be closed include domain specific languages and data standards, and formal semantics to encourage and enable model interoperability and reuse [Bergenthal, 2011].

6.2 System modelling languages, frameworks and tools

MBSE approaches address different system modelling formalisms, languages and frameworks in order to provide and control a system model enabling to represent all aspects of a system at all levels of abstraction across the lifecycle and across disciplines. This section gives an overview of existing and recognized object-oriented languages that intend to answer these requirements as well as the frameworks that have been developed based on these formalisms.

6.2.1 Object-oriented System modelling languages

6.2.1.1 The Object-oriented approach

Object-oriented modelling is a modelling paradigm that has its inception in computer programming and also known as object-oriented programming. The object-oriented paradigm intended to help programmers to handle the complexity of a system and related problems by considering the system not only as a set of hierarchical functions that need to be performed, but above all, as a set of related, interacting objects. Each object represents some entity of interest in the system being modelled, and is characterized by its class, its state (data elements), and its behaviour. Various models can be created to show the static structure, dynamic behaviour, and run-time deployment (instances) of these collaborating objects. The object-oriented programming and its fundamental concepts (abstraction, encapsulation, inheritance and polymorphism) are now widely proven. The object-oriented approach has become a key technology when one seeks to develop complex software or system which functionalities and behaviours might evolve continuously.

However, the object approach is less intuitive than the procedural or functional approach. It is easier for the human mind to break a problem as a hierarchy of functions and data than in terms of objects and interaction between these objects. Therefore in order to guide the system designer to use object-oriented approach and concepts, an object-oriented language and formalism was required. This language must enable to represent abstract concepts (e.g. graphically), reduce ambiguity (speak a

common language with a precise vocabulary), be independent of existing programming object-oriented language (like C, Java or Python) and enable analysis (simplify the comparison and evaluation of solutions). Between 1970's and 1990's, with the advent of software engineering, many researchers have developed object-oriented approaches. However, only three methods have truly emerged: The OMT method Rumbaugh [Rumbaugh et al., 1990], the BOOCH'93 method [Booch, 2006] and the OOSE [Jacobson, 1992]. In 1994, Rumbaugh and Booch (joined in 1995 by Jacobson) have joined their efforts to develop a unified approach incorporating the advantages of each of the above methods. This unified approach was subject to the Object Management Group (OMG) - a group of experts developing computer industry standards for validation: the object-oriented **Unified Modelling Language** (UML) was born. UML is a visual language for specifying, constructing, and documenting the artefacts of systems. It is a general-purpose modelling language that can be used with all major object and component methods, and that can be applied to all application domains and implementation platforms [Booch et al., 2000].

The version 2.0 of UML addresses the problems of modelling architectures. It enhances the capability for modelling hierarchical structure and behaviour. [OMG, 2010b]. It was the first step towards an object-oriented language dedicated to system engineering. However it did not allow the modelling of flows on links. Moreover, links to requirements, parametric equations and others were still not addressed. Therefore, the Object Management Group (OMG) developed **SysML** for systems engineering support, extending some existing modelling diagram functionalities and adding parametric and requirement diagrams functionalities. The following Figure 40 shows the various types of diagrams that are handled by the UML language and the ones added by SysML.

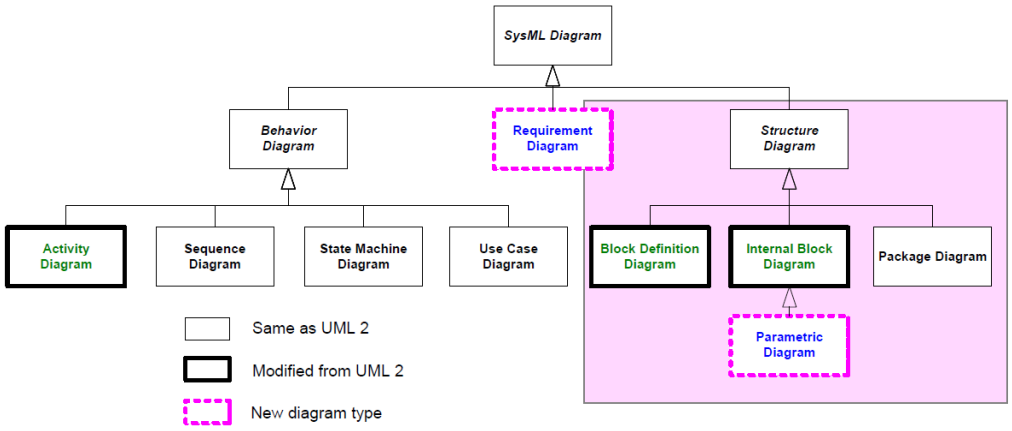


Figure 40: UML and SysML diagrams taxonomy

6.2.1.2 SysML

The Object Management Group OMG has developed SysML: "a general-purpose graphical modelling language for representing systems that may include combinations of hardware, software, data, people, facilities, and natural objects". SysML supports the practice of model-based systems engineering (MBSE) that is used to develop system solutions in response to complex and often technologically challenging problems [Friedenthal et al., 2011].

In the **structure diagram**, the physical system architecture is represented by **block definition diagrams** and **internal block diagrams**. A block definition diagram describes the system hierarchy and system/component classifications. The internal block diagram describes the internal structure of a system in terms of its components (blocks), ports, and connectors [OMG, 2010a].

Blocks are modular units of system description. Each block defines a collection of features to describe a system or other element of interest. These may include both structural and behavioural features, such as properties and operations, to represent the state of the system and behaviour that the system may exhibit. Blocks provide a general-purpose capability to model systems as trees of modular components. Therefore they enable to represent multiple hierarchical levels of the system on a same diagram and specify links between different breakdown levels of the system (like behavioural links at interfaces) [OMG, 2010a].

A **port** is an interaction point between a block or part and its environment that is connected with other ports via **connectors**. Specifying such ports on system elements allow the design of modular reusable blocks, with clearly defined interfaces. SysML provides three types of ports [OMG, 2010a]:

- Standard Ports are used to specify service oriented peer-to-peer interaction which is typical for software component architectures. Standard ports typically contain operations that specify bidirectional flow of data, so they are typically used in the context of peer-to-peer synchronous request/reply communications.
- A flow port specifies the input and output items that may flow between a block and its environment. Flow ports are interaction points through which data, material, or energy can enter or leave the owning block. The specification of what can flow is achieved by typing the flow port with a specification of things that flow.
- Item flows represent the things that flow between blocks and/or parts and across associations or connectors. Whereas flow ports specify what “can” flow in or out of a block, item flows specify what “does” flow between blocks and/or parts in a particular usage context. This important distinction enables blocks to be interconnected in different ways depending on its usage context.

The **behaviour diagram** enables the sequence of events and activities that the system must execute. The requirements diagram captures requirements of the client to the model and guides the whole design work to provide unambiguous traceability between the requirements and system design [Paredis et al., 2010]. The **requirement diagram** provides a bridge between typical requirements management tools and the system models. The **parametric diagram** is dedicated to modelling networks of constraints on system properties to support engineering analysis, such as performance, reliability, and mass properties analysis. Parametric diagrams include usages of constraint blocks to constrain the properties of another block. The usage of a constraint binds the parameters of the constraint to specific properties of a block that provide values for the parameters [OMG, 2010a].

In the scope of our research work and in view to model mechanical systems and to ensure continuity of information between system design and analysis models across multiple domains, we focus our attention on the structural diagrams. However, according to [Paredis et al., 2010], the behavioural, structural diagrams and the requirements diagram together provide not only an integrated view but also multiple views of a system. These multiple views can be maintained consistently due to the semantic underpinning of the modelling language. Moreover in section 6.3 (Object-Oriented System modelling for design-analysis data integration), a research work proposing to use parametric SysML to ensure associations the behavioural parameters of a mechanical analysis model to the related CAD model parameters is mentioned.

6.2.1.3 Modelica

Modelica is a non-proprietary, object-oriented, equation based language to conveniently model complex physical systems containing, e.g., mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents [Modelica, 2010]. The behavioural model is based on ordinary and differential algebraic equation (OAE and DAE) systems combined with discrete events, so-called hybrid DAEs. Such models are ideally suited for representing physical behaviour and the exchange of energy, signals, or other continuous-time or discrete-time interactions between system components. The Modelica Language is defined and maintained by the Modelica Association which publishes a formal specification but also provides an extensive Modelica Standard Library that includes a broad foundation of essential models covering domains ranging from (analogical and digital) electrical systems, mechanical motion and thermal systems, to block diagrams for control [Schamai et al., 2009]. A large number of Modelica simulation environments are already available in commercial and free applications; the list is provided here: https://www.modelica.org/tools.

Modelica models are similar in structure to SysML models in the sense that Modelica models consist of compositions of sub-models connected by ports that represent energy flow (undirected) or signal flow (directed). Figure 41 Shows the same system (a mass suspended by a spring) represented in Modelica (on the left side) and in SysML.

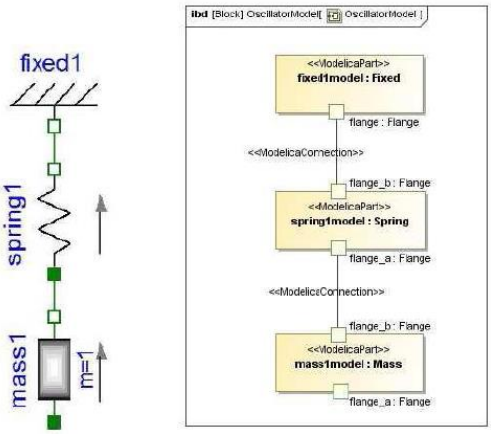


Figure 41: Formalism difference between Modelica (on the left) and SysML (on the right)

OpenModelica is an open-source Modelica-based modelling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium [Fritzson et al., 2006].

6.2.1.4 SysML-Modelica transformations

SysML and Modelica are two complementary languages, integrating the descriptive power of SysML models with the analytic and computational power of Modelica models provides a capability that is significantly greater than provided by SysML or Modelica individually. According to the overview proposed by [Paredis et al., 2010], there are three main methods related to the transformation between SysML and Modelica up to now.

[Pop et al., 2007] propose to convert SysML diagrams to Modelica simulation and to provide a SysML/UML view of Modelica for documentation purposes and language understanding as well as extending Modelica with additional design capabilities. The translation between Modelica and SysML models is done via XMI (XML Metadata Interchange) which is an Object Management Group (OMG)

standard for exchanging metadata information via Extensible Mark-up Language (XML). They have created a UML profile called **ModelicaML** that reuses some of UML and SysML diagrams while adding several new diagram types such as the equation diagram, and the simulation diagram.

More recently, [Schamai et al., 2009] has extended this work. As Modelica models are similar in structure to SysML models, authors represent Modelica models with related ModelicaML diagrams. As such, Modelica structures are represented by Modelica structural diagrams, Modelica conditional equation or algorithm statements are modelled using ModelicaML behaviour diagrams. Requirements diagram in ModelicaML enables traceability between textual requirements and design artefacts, and supports impact analysis when requirements and/or the model change.

Since SysML models and Modelica models can both be represented graphically, [Kindler&Wagner, 2007] propose to simplify model transformations to graphical transformations, using Triple Graph Grammars. Triple Graph Grammars (TGGs) are a technique for defining the correspondence between two different types of models in a declarative way. The power of TGGs comes from the fact that the relation between the two models cannot only be defined, but the definition can be made operational so that one model can be transformed into the other in either direction; even more, TGGs can be used to synchronize and to maintain the correspondence of the two models, even if both of them are changed independently of each other.

6.2.2 MBSE modelling frameworks and tools

Most of MBSE methodologies mentioned in 6.1.2 do not have any process framework tool to support the methodology since they were created as tool- and vendor-neutral methodologies. For the ones having a supporting framework tool, it is necessary to distinguish:

- tools dedicated to generic system modelling engineering including all enterprise systems modelling applications such as enterprise, software, hardware or product architectures modelling tools,
- tools only dedicated to software engineering process,
- model-based and object-oriented applications integrated in domain-specific engineering and analysis applications or within PLM platforms.

The scope of this PhD concerning improvement for modelling mechanical physical and behavioural systems and for ensuring continuity of information between system design and analysis data/models across engineering domains, this section only surveys a subset of these system modelling tools that can be relevant for this scope.

The IBM RUP-SE methodology is supported by a RUP-SE plug-in in the **Rational Method Composer** (RMC) tool integrated in the IBM Rational suite of tools aiming at supporting analysis, modelling, design, and construction with a software development focus [Estefan, 2007]. Most of these tools mentioned, including RMC, are supported on the Eclipse open source platform managed under the auspices of the Eclipse Foundation.

IBM Rational Rhapsody tools suite family encompass **Rational Rhapsody Designer** supporting the MBSE Rhapsody methodology using SysML for visualization of complex requirements and model execution for early validation of requirements, architectural trade off analysis and mitigation of project risks [INCOSE, 2008].

CORE (Vitech Corporation) is a system modelling environment that includes integrated modelling capabilities to assess and control design and program risks. The aim is to link “all” elements of the system through a central model with an emphasis on visualizing system development risk drivers. Main

CORE's capabilities include integrated requirements engineering management tool, system architecture modelling and development tools based on SysML to assign functionality and requirements to physical architecture models. A model repository to track of all the necessary components and subsystem elements is also provided to ensure change propagation from one modified diagram to all related impacted views and executable behaviour models based on system logical diagrams and information flows diagrams to demonstrate system functionality and performance [INCOSE, 2008].

Cradle (developed by 3SL) is presented as a systems engineering environment supporting the entire system lifecycle. This includes requirements management, process definition and business analysis, system architecture definition and assessment, high-level and component design, test management, V&V. It focuses on gathering and crossing information from all of these activities in order to ensure traceability and coverage analyses across the entire system lifecycle [3SL, 2013].

ArKItect™ (developed by Knowledge-Inside) is an object-oriented environment for modelling multi-disciplinary systems and specifications. Main functionalities enable to describe functional and physical architectures, allocate requirements to functions, construct validation plans, and follow-up models evolutions. The principle of ArKItect is to use and synchronize these different representations and ensure their consistency within a single system model. A hierarchical types definition system enables to customize the graphical representation and to apply arKItect ™ to "any" kind of technical domains. arKItect™ Designer and Developer modules permit to define graphic meta-models and deploy them. A relation matrix editor allows the user to define his own objects and flows, to assign attributes to them and to define their composition rules. **System Engineering Essentials** is a specific provided meta-model that enables to develop your systems following standard system design stages. Knowledge Inside has developed two domain-specific extensions of System Engineering Essentials: a Mechatronics module and a Safety module [Knowledge Inside, 2013].

Sodius has developed **MD Workbench** that focus on models interoperability, enabling the creation of new tooling connectors and encapsulation complexity of direct connection to authored data; providing a kind of adaptable model hub architecture. For instance it provides a variety of translators between SysML applications, including diagrams, a number of specialized modelling tools for Space and Defence, meta-models to gather interface data coming from various sources and connectors between UML/SysML and non-UML-based tools such as DOORS or Matlab-Simulink [Sodius, 2013].

Thales has developed its own MBSE methodology named **Arcadia**. The toolset supporting ARCADIA is named ORCHESTRA and runs over Eclipse. The heart of **ORCHESTRA** is an architecture modeller/checker called **MELODY ADVANCE**. MELODY ADVANCE provides a modelling environment based on UML/SysML but customized with engineering semantics. It enables to enrich and extend ARCADIA basic concepts (so called "meta model") for specific domains and specialty engineering, to customize existing diagrams and create new kinds of diagrams (with a Domain Specific language) for dedicated analysis, to define model analysis and check rules, as needed for each viewpoint, to develop multi-viewpoint compromise analysis tools and to reuse design artefacts in appropriate context and capitalize decisions (e.g. reuse libraries and checking viewpoints, architectural patterns management) [Voirin, 2010].

Phoenix Integration has developed the **Phoenix Integration Software Suite** for meeting the needs of Simulation Driven Design. The aim of the Phoenix Integration platform is to create and maintain a library of modelling and simulation tools and simulation workflows, automatically execute the workflow, leverage computing resources to perform trade studies and ask "what-if" questions, and archive, manage, and share the resulting data and meta-data. The Phoenix product suite is divided into four primary applications:

- ModelCenter: process integration environment enabling to create simulation workflows, perform trade studies, and analyze, visualise the results;

- Analysis Server: a light-weight server tool for remotely executing analysis tools;
- CenterLink: web based application for executing collaborative ModelCenter simulation work-flows;
- AnalysisLibrary: shared drive replacement for managing simulation file content and enhancing re-use.

As shown in Figure 42, the **Phoenix** Software Suite is designed so that it can be flexibly deployed in conjunction with PDM systems. In this scenario, the simulation management framework acts as a low overhead "simulation sandbox" for the engineering team. Engineers check requirements and geometry out of one or more PDM systems, utilize the simulation framework to run analyses. After a final result has been achieved, updated geometry and final analysis results can be checked back into the PDM systems. In addition to the core capabilities, ModelCenter can also be upgraded with additional CAD and CAE plug-ins enabling a direct integration of ModelCenter with CAD tool (automating the import of CAD design parameters for use as variables in ModelCenter) and CAE tools (to interoperate and automate CAE tools such as MSC Nastran, NX Nastran, ANSYS, Abaqus, LS-Dyna, and MSC Adams) [Woyak, 2010].

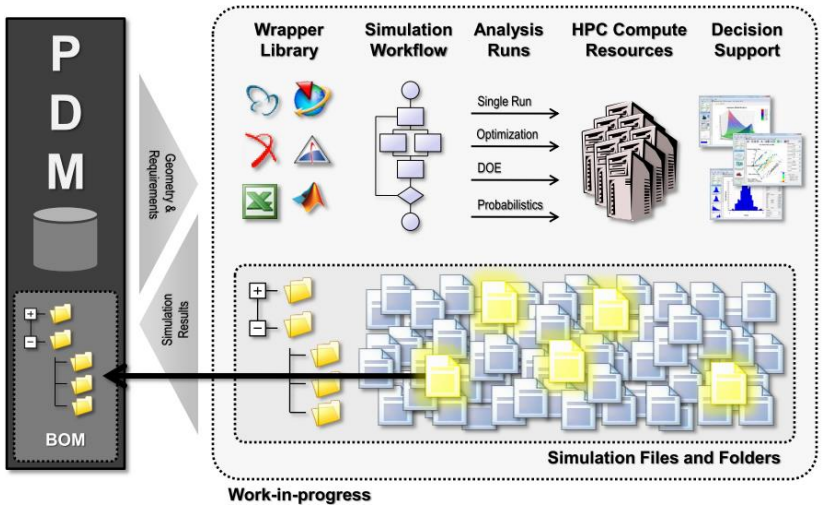


Figure 42: Phoenix Integration applications and PDM systems interoperating principle from [Woyak, 2010]

The **CATIA/SIMULIA V6** platform developed by Dassault Systèmes (DS) is now based on the unified RFLP approach (Requirements, Functional, Logical and Physical Design). This platform provides an object-oriented system architecture modelling framework with full traceability between functional, requirements and logical blocks, ports, connectors and flow items. V6 virtual execution platform enables to execute and analyze system models, mixing dynamic and state logic behaviours. CATIA Systems Logical 3D Architecture brings 3D to logical systems for space reservation and pathways connection. It provides 3D modelling tools for systems architect to define and investigate several 3D layout alternatives early in the product design process. Integrated with DS Digital Mock-Up and "Knowledgeware" products CATIA Systems Logical 3D Architecture allows the validation of the 3D architecture of logical systems with respect to installation requirements.

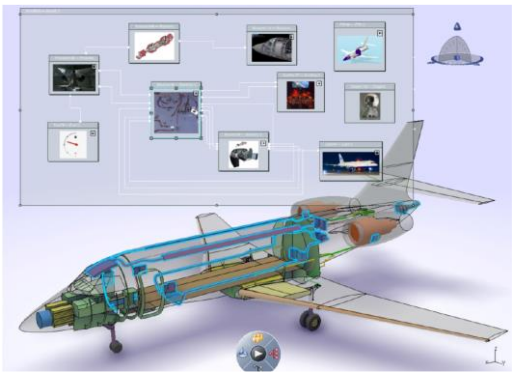


Figure 43: CATIA/SIMULIA V6 RFLP framework integrated with 3D CAD DMU environment

In order to simulate logical system architectures, DS PLM platform has integrated **Dymola**; the foundation technology for "CATIA V6 Dynamic Behaviour Modelling", based on Modelica. Dymola is a modelling and simulation environment which aim is to simulate the dynamic behaviour and complex interactions between systems and sub-systems. It provides ready-to-use model libraries and the structure based on components, including 3D and their behaviour for many engineering fields (such as mechanical, electrical, thermodynamic, hydraulic, pneumatic, thermal and control systems). In order to simulate 3D CAD models, SIMULIA provides simulation capabilities and seamless integration into their CAD environment using the DesignSight and CATIA Analysis products. Coupled with Isight Execution Engine (formerly Fiper) it permits to combine multiple cross-disciplinary models and applications together in a simulation process flow, automate their execution across distributed compute resources, explore the resulting design space, and identify the optimal design parameters subject to required constraints.

Teamcenter 9 (developed by Siemens PLM software) offers the same kind of system engineering approach with Systems Architect. This application enables to create systems-level product architecture by capturing multiple product views, including views of the product's features, functions, physical content and logical hierarchy. Although logical architecture are modelled with an integrated Visio module that still present some modelling limitations compared to SysML or Modelica formalisms, Teamcenter also provides full traceability between functional, requirements and physical design artefacts. In this platform there is also the will to provide traceability between CAD and CAE in specific defined design context allowing to opportunities for re-use, as well as cross-discipline and cross platform trade-off integration.

SLIM (developed by InterCAX) is an integrated software platform for systems lifecycle management. It is envisioned to provide capabilities that combine the strengths of model-based systems engineering and product lifecycle management (PLM). It uses SysML as the front-end for multi-disciplinary teams to collaboratively develop a unified, coherent representation of the system from the earliest stages of development. The system model (in SysML) can 'co-evolve' with the associated domain-specific models, such as Computer-Aided Design and Engineering (CAD/CAE) models. Relationships between the system model and the domain-specific models can range from qualitative dependency relations to quantitative causal parametric relations which are executable on-demand for seamless model traceability and interoperability. Figure 44 shows the conceptual architecture of SLIM.

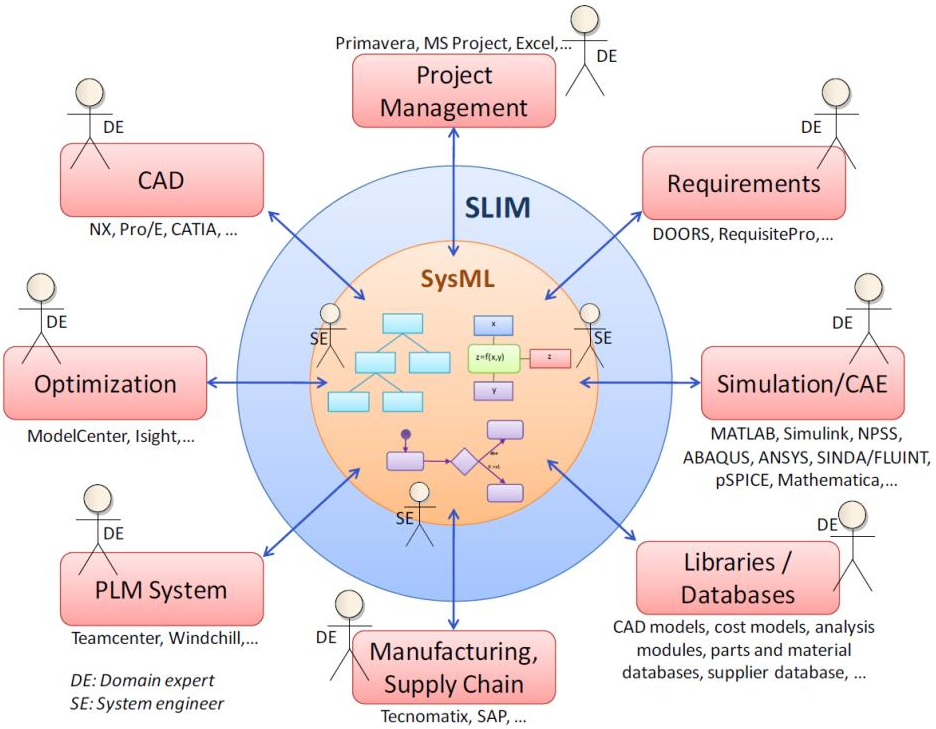


Figure 44: Conceptual Architecture of SLIM from [Bajaj et al., 2011]

6.3 Object-Oriented System modelling for design-analysis data integration

In the literature, two complementary approaches enhance the use of object-oriented modelling techniques to support Design-Analysis and more specifically CAD-CAE data integration. The first one consists in using the object approach (use of blocks, ports and connectors) to link design and behavioural models parameters in order to automate the composition and the update of the models. The second consist in using ports and connectors to enrich interfaces and interaction information and to be able to change components definition without modifying the interfaces definition.

The **Composable Simulation Project**, originally developed for mechatronics systems at the Institute for Complex Engineered Systems of Carnegie Mellon University, is based on the idea that system level simulations can be automatically generated from individual components from a CAD system. This allows for systems to be simultaneously designed and simulated [Sinha et al., 2002]. This approach proposes to design CAD models and "**Composable simulation models**" in an object-oriented formalism. The technology permits to define a simulation models hierarchy and multiple models (models fragments) can be associated with a single system component. These models are organized so that model fragments can be easily reconfigured (through composition and instantiation) to suit a particular simulation context and hence enhance re-use by selecting the appropriate model for the current phase in the design process. Model parameters are automatically extracted from the CAD geometry and material properties [Paredis et al., 2001] [Sinha et al., 2002]. The Composable Simulation Project is supported by **Reconfigurable Models** and **Component Libraries**.

A Reconfigurable Model is a system representation based on interface and implementation. Interface is used to describe the interaction through ports and implementation described the internal behaviour of a system. The Component Library is a set of reconfigurable models for use by the de-

signer/analyst. The goal is to achieve different configurations of the same system by altering the different implementations while keeping the interfaces constant. Reconfigurable component models provide a mechanism for describing system changes to both structure and parameters. Multiple configurations and simulation instances can be achieved by changing parameters and reconfiguring system components, known as composition and instantiation [Diaz-Calderon, 2000].

The component library permits to store and re-use a set of reconfigurable models. Two kinds of models are present in the library: system component models and component interaction models. The idea is to organize and classify the models in an intuitive manner so that the designer can easily retrieve and compose the appropriate analysis model. Additionally, a component may be an abstract simulation model, where the structure is defined, but the system parameters are not yet instantiated. The intent was to permit designers to progress from highly abstract representations for first-run simulation to detailed components for final design [Diaz-Calderon et al., 2000].

To achieve tight integration of design and analysis, design models should support the creation of composable simulation/behavioural models. Simulation models should also support design model views. Sinha, et al. developed a design environment using the component library and where the simulation model and the design model can be created simultaneously [Sinha et al., 2000, Sinha et al., 2001a, Sinha et al., 2002]. As shown on Figure 45, in this environment, a component is a modular design entity with a complete specification describing how it may be connected to other components in a configuration. Behavioural models capture the mathematical description of the physical and informational behaviour of a component. Behavioural models can also be composed out of other behavioural models through the port-based modelling paradigm [Sinha et al., 2002]. A component object can contain multiple behavioural models with different levels of detail. This technology is based on encapsulations: an object can only be accessed through its public interface, which is independent of the underlying implementation.

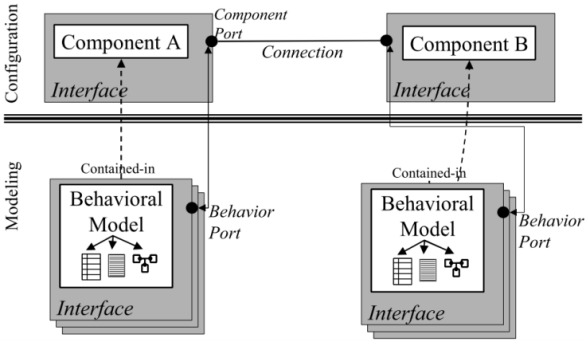


Figure 45: Configuration of components and interfaces and selection their behavioural models from [Sinha et al., 2002]

Authors apply the same principle for modelling mechatronics systems by making a clear distinction between the physical interactions of an object with its environment (interface) and its internal behaviour (implementation) [Sinha et al., 2002]. This allows modelling a system by composing and connecting the interfaces of its sub-systems, independently of the future implementations of these subsystems. In this environment all interactions between components are mediated by ports. The high-level component ports in the component interface are related to the ports of the behavioural model interfaces encapsulated in the component. The interaction model then becomes a container for this set. The container holds all the possible behavioural models that can be used to represent this interaction. The container is populated with interaction models stored in a library for re-use. The parameters of the interaction can be inferred by geometric reasoning on the CAD data in each component. In order to organize and maintain the space of all possible interaction models, and to support evolution of the

design, authors propose port and interaction model taxonomies (see Figure 46 below). Within each of these categories, models are classified by the physical domains that they represent. The choice of a particular model from the container depends on the nature of the simulation experiment that is being performed [Sinha et al., 2001a, Sinha et al., 2002].This type of design and analysis integration has been implemented in electrical CAD (ECAD). However, most commercial mechanical CAD applications do not support this type of integration.

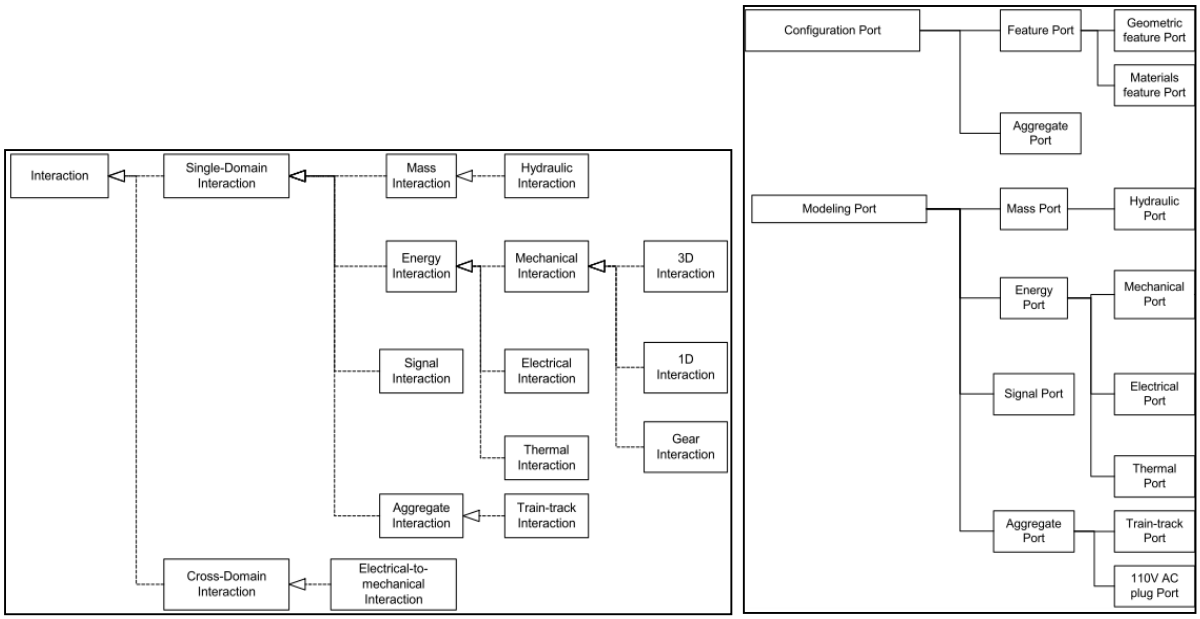


Figure 46: Interactions (left) and ports (right) taxonomies proposed by [Sinha et al., 2001b]

The **Multi-view Representation Architecture** (MRA), developed at the Engineering Information System Laboratory (EISLab) at the Georgia Institute of Technology, is addressing the gaps between CAD and CAE tools. The methodology is based on knowledge patterns that naturally exist in engineering analysis processes and on explicit design-analysis models associativity. The goals are to automate routine analyses, ensure design and analysis associativity and of the relationships among the models, and to provide and re-use analysis models throughout the life cycle of the product. The MRA attempts to bridge the gap between design and analysis based on four building block constructs [Peak et al., 1999]:

- The Solution Method Models (SMM) represents solution-specific methods combining inputs, output, and control for a single type of analysis solution. It is a wrapper that serves as tool agent to provide information on what solution tool to use, the inputs to the tool, the control for the tool, and how to retrieve results from the tool.
- Analysis Building Blocks (ABB) represent engineering concepts that include engineering semantics and are independent of the SMM. Analysis systems are assemblies of ABBs to represent a particular model. ABBs are constructed utilizing constraint graphs and object-oriented techniques. ABBs use transformation operators to be linked with SMMs. The SMM instance is created from inputs based on the ABB. The nature of ABBs allows for different solution methods to be used.
- Product Model (PM) is the product data model representing all data associated with the product over its lifecycle. In addition to CAD and CAE geometric data the PM model encompass design information items such as loading and boundary conditions. When it was created the PM was one of the first steps towards an integration of simulation data within PDM systems: a sketch and a basis for future SDM systems. The idea was to capture idealizations and simplifications rules applied on analysis models in the PM in order to be re-used. Product Model-Based Analysis Models (PBAMs) contain the linkages between the PM and the ABBs. PBAM

analysis models are particularly useful for routine analysis where libraries of ready-to-use analysis modules are created and available for use in later analysis activities.

Later, Peak et al. also introduce the **Composable Objects** (COB) representation which is based on object and constraint graph concepts allowing capturing diverse multi-fidelity models and their fine-grained relations. Later, Peak et al have transformed the MRA patterns and representations into COBs that can be implemented in SysML. Within a MRA context applied on a flap linkages part, authors have demonstrated the usage of parametric SysML and COBs at component level, linking the behavioural parameters of a mechanical FEA model to the related CAD model parameters [Peak et al., 2007a, Peak et al., 2007b] (see Figure 47).

Remaining challenges are to manage this CAD-CAE integration at assembly level and establish relations at different levels of system decomposition as well as to ensure the continuity of information between cross-domain models.

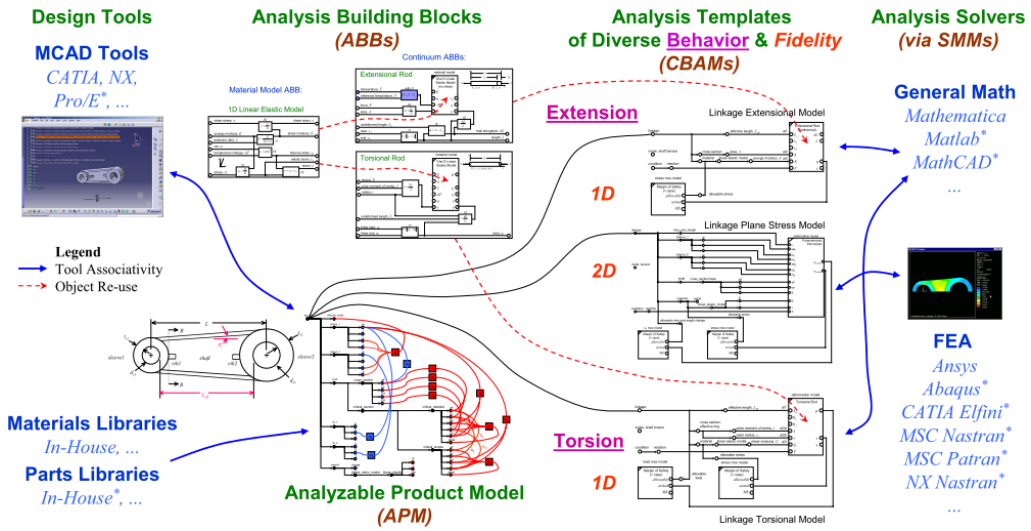


Figure 47: COB/MRA-based panorama for CAD-CAE interoperability from [Peak et al., 2007b]

The **MOSAIC** (Integrated modelling and simulation of physical behaviour of complex systems) project, based on research by K. Andersson and U. Sellgren at the Royal Institute of Technology in Sweden, aims at developing of an object-oriented model of behaviour of the product. Toward this end, a general product model applicable to the entire product development process and a prototype system to support design and simulation of complex products have been developed [Andersson, 1999]. The prototype MOSAIC system consists of a process model, object model, libraries of requirements and analysis models, system models, methods for validation, and methods for translating requirements to technical specifications. Figure 48 shows the activity and data chart proposed by Anderson to represents the data associated with design and analysis activities.

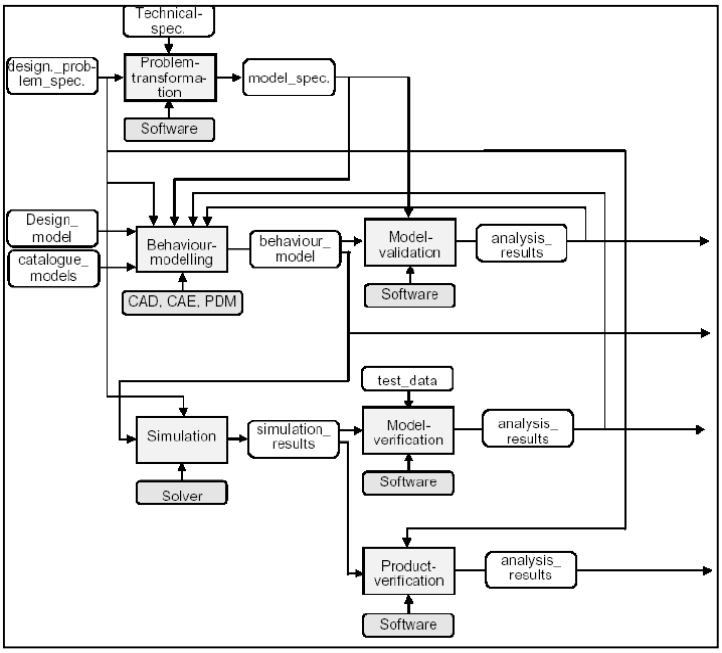


Figure 48: Design- Analysis activity and data chart from [Andersson, 1999]

The approach of the MOSAIC system enables the product to be divided into a number of subsystems to be analyzed. Each system can be characterized by what is within its boundaries and how it interacts with other systems. Interfaces between systems are described by mating features and interface features. Mating features are used to characterize the position of the connected systems. Interface features characterize the connection between the mating features. In other word, mating features are what is connected between two systems and interface features are how the two systems are connected. Because connections consist of both mating classification and interface classification, the systems are easy to modify. Multiple design alternatives can be developed by changing the interface connections [Andersson&Sellgren, 1998]. Interfaces have characteristic properties that cannot be directly derived from the related mating features. [Sellgren, 1999] highlights the need to rely on a modular model architecture that enables configuration of systems models from a stored library of sub-models and interface models. Later Sellgren proposes a model-based and feature-based interface information model as extension and improvement of PDM data models [Sellgren, 2006a, Sellgren, 2009] (see section 8.1.2).

6.4 Synthesis of current MBSE gap and limitations

The dominating issue of the dysfunction in MBSE is the lack of connection between models and model elements, which appears not only between different languages but also within one language. According to [Herzig et al., 2011], consistency implies an absence of contradictions. Authors classify consistency in two groups:

- Internal consistency and external consistency. Internal consistency problems relate to axiomatic systems that are well understood (e.g. logic systems and mathematics). Based on these systems we construct modelling languages. Models that are internally consistent do not violate the axioms and rules of the underlying formal system – they are theorems of the system.
- External consistency imposes an additional constraint, namely, that the model be true to reality.

Finally authors conclude that checking for external consistency issues is impossible to achieve, since it would require perfect knowledge about the processes occurring in nature; And it is impossible to say with certainty whether a set of models is consistent. Instead, we can only detect specific types of inconsistencies within the bounds of a formal system.

Concerning SysML and Modelica formalisms several limitations have been identified:

- SysML structural architecture is described in terms of blocks but it is not possible to highlight a difference between hardware and software components.
- The synchronization between SysML and Modelica models, while defined with the help of stereotypes in SysML, are not distinguishable within Modelica anymore. This makes the automation of the feed-back of simulation results into the system description challenging and still not standardized [Votintseva et al., 2012].
- When simulation models need to be modified another limitation for the analyst is to keep simulation models compliant with the original model in SysML.
- As most multi-domain systems are designed in increasingly large teams, component interfaces are often set up at early design stages, and later can be modified corresponding to new requirements or other changes in the environment. This implies rework on the existing models.
- To fully synchronize SysML structural models with simulation models (behavioural models), it has to be decided which information should be modelled within the system description language and the simulation model. A system architect must be able to identify the goal of the simulation at different development phases and specify simulation relevant attributes in a non intrusive way. Therefore, SysML models needs to be fed with the appropriate data sets relevant for the generation of a meaningful simulation model.

In [Bajaj et al., 2011], authors underline two major gaps to close in order to ensure a MBSE design-analysis integration approach.

The first gap concerns the lack of model-based continuity of system design and simulation activities from the early mission design phases to the later design phases. This gap exists because the modelling and simulation tools used to create system models (design and analysis) are different in early mission phases versus the later mission phases.

The second gap concerns disconnects between design and analysis/simulation models in different design phases, such as between conceptual system design models and math-based analysis models in early design phases or between CAD and CAE models in detailed design phases. In general, the second gap manifests in heterogeneous model transformations beyond design and analysis, such as between requirements and structure, logical structure and physical structure, and structure and behaviour [Bajaj et al., 2011].

Majority of identified MBSE tools provide system modelling architectures capabilities (whether functional, structural or behavioural). Some of them also provide capabilities for workflow simulations or simulations of 1D behaviour models. Many of them also provide a system meta-model ensuring cross-domain models consistency and integration. However, very few provide capabilities to make these system architectures, process or other simulation models, interact with PDM systems and DMU environments. The reason is that these tools have been mainly developed for software system engineering applications, which do not require interacting with CAD or CAE models. [Sinha et al., 2001a] propose that the object-oriented programming design methodology can be applied to mechanical systems modelling. According to authors, the object-oriented modelling approach, as leveraged from the software development domain, is a step in the natural progression of modelling mechanical systems.

For mechanical system design, the DMU has become the main federating environment for sharing 3D digital data within a collaborative context. In [Alemanni et al., 2011] authors even underlines the predominant role of CAD-DMU environments and data/models in a MBSE approach applied to digital product design. Nowadays, design, integration and verification/validation activities are performed through the use of CAD and CAE tools. In an MBSE approach these tools and related models should populate, interrogate and exploit the system model in order to identify, structure, retrieve, share, disseminate and visualize product engineering data. Therefore, the majority of existing MBSE tools needs a technological leap to offer new system modelling capabilities exploitable in the field of digital mechanical product design and enabling to interoperate efficiently with CAD and CAE tools.

Chapter 7: The DMU as the backbone of Model-Based and Simulation-Based Mechanical Product Design

In a model-based approach, the design phase of a product life cycle aims at creating a complete DMU including all information on the product coming from multiple points of view: functions, components, form features, materials, multi-physical behaviours [Shah, 1991], [Tichkiewitch&Véron, 1997] [Yan, 2003]. Therefore, in a MBSE process applied to mechanical systems, and as mentioned in the DMU-related industrial challenges, the DMU must be the reference of the product definition and the BOM must be generated from the DMU. Doing so, inconsistencies between these two product definitions are avoided. However this requires establishing very strict and standardized rules and methods between all co-designers and partners involved in the construction, enrichment and exploitation of the DMU.

Another mentioned DMU-related challenge is to integrate behavioural simulation data and processes with the DMU and providing the Behavioural-Digital Mock-Up. Although the bi-directional interfacing concept between the BMU and its associated DMU is still not defined in the literature, there have been initiatives to ensure continuity and traceability of information between CAD models, their assemblies in DMUs and the simulation data and activities that use this design definition support as input.

7.1 DMU: the multi-view point product definition referential

The DMU has been wrongly considered as an environmental "catch-all" to which many people and trades involved in the product development seek to cling to [Drieux, 2006]. In theory, it is generally seen as a reference object of the product definition. Several authors have performed analyses of the deployment and exploitation of DMU environments in the aircraft industry; demonstrating the potential benefits of using it as the product definition reference and underlining the related technical challenges in a collaborative and distributed environment [Nguyen Van, 2006] [Garbade&Dolezal, 2007] [Guyot et al., 2007] [Dolezal, 2008] [Toche et al., 2012].

Unfortunately, in practice this is still not the case because of the inconsistencies issues mentioned in 3.3.4. In order to be considered and used as the product definition referential, it is essential for a DMU environment to enable the representation of a system from multiple viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail, fidelity and abstraction. This enhances the application of the **multiple product views** concepts – as it has been defined from an engineering design perspective [Rosenman&Gero, 1996] [Léon, 1999] – on the DMU product representation.

Along the product life cycle, group of stakeholders involved in the development of the product addresses the product from a particular viewpoint. The corresponding description of the product from a particular viewpoint characterizes a product representation, i.e. a model in terms of elements exploitable in the field of knowledge of the corresponding stakeholders [Hamri et al., 2008]. **A product view** defines the link between a product representation and the activity or process (performed at least by one stakeholder) that use or generate this representation as respectively input or output. However, the concept of DMU conveys different meanings and has not been clearly defined with regard to the concept of product view.

As shown in Figure 49, [Drieux, 2006] makes a clear distinction between the DMU used as a product definition reference and the DMU used in downstream applications. He also distinguishes the DMU data providers (designers) who build the referential DMU and the DMU data consumers that use whether the referential DMU whether a derivative and adapted DMU representation.

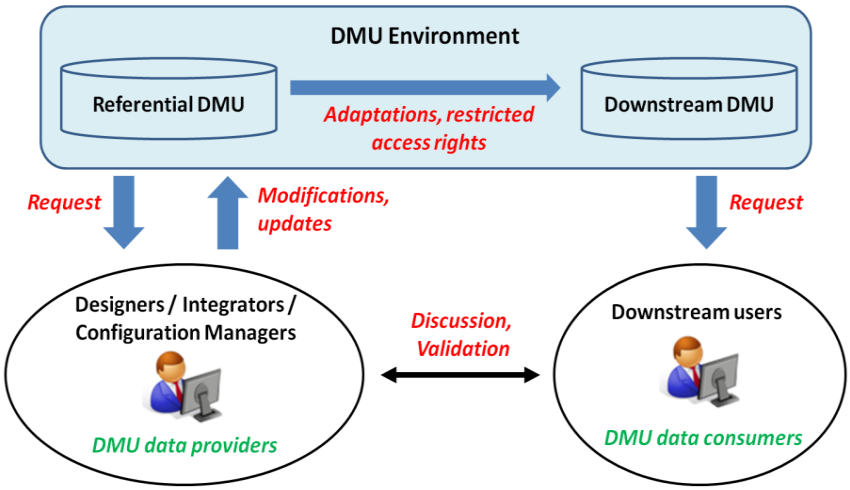


Figure 49: Differentiation of DMU stakeholders roles re-created from [Drieux, 2006]

Thus the author introduces the concept of Downstream DMU (DDMU) referring to the product views where simulations activities involving the product/component shapes may take place. From the product development process point of view, the product views other than the Design view performed by the engineering office and producing the DMU can be regarded as tasks located downstream with respect to the Design view, hence the name of the digital models that can be produced through the simulations operated by these Downstream product views . Indeed, DDMU designates digital product models derived either from the DMU or from DDMUs previously generated. Figure 50 provides a schematic view of the main processes attached to a DDMU processing. The task flow between the design process and the downstream process illustrates the interactions between DMU and DDMUs over time according to the progress of the Design process.

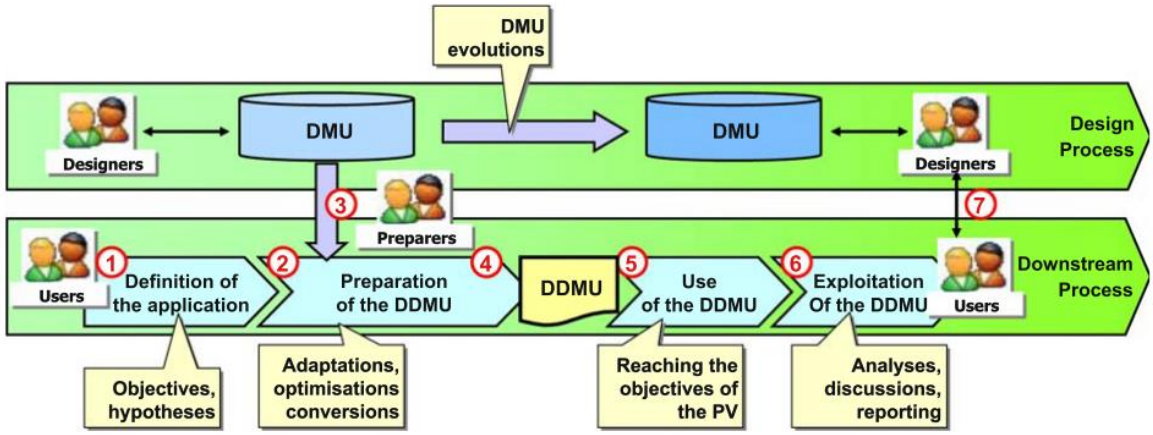


Figure 50: Structure of a DMU processing to produce a DDMU for a given product view from [Drieux, 2006]

All these transformations/adaptations must be performed consistently with regard to the domain-specific needs of the activity using the "DDMU". In addition to mechanisms of adaptation, the process transforming a DMU into a DDMU might integrate enrichment mechanisms. For this PhD dissertation the focus of attention are the adaptations required to provide the appropriate DMU content and structure to be used for large integrated assembly FEA. In such a context, three kind of required transformation to pass from a DMU to a DDMU have been identified:

- **DMU shapes transformation**
- **DMU enrichment in terms of functional and topological components' interfaces**
- **DMU Structural transformations**

7.2 DMU transformations for integrated assembly FEA

To reach the needs of large assembly simulation models, improvements in processing DMUs are a real challenge in aircraft companies. Only a few and recent research works highlight the DMU potential for being the backbone of design-simulation loops and to be adapted for domain-specific engineering needs and especially for simulation needs. Nevertheless, DMU is often used to prepare the structural analysis of a whole assembly or to generate a fluid domain for thermal and CFD calculations. Therefore adapting DMUs represents a very time-consuming and tedious effort due to gaps introduced in section 4.1.1. This section gives an overview of current research works that aim at speeding-up the required DMU enrichment and transformations operations so that DMUs can serve as reliable input for FEA. According to [Mocko&Fenves, 2003], CAD-FEA integration research can be categorized into two focus groups:

- Microscopic approaches deal with automatic mesh generation, model simplifications and idealizations, loading boundary condition required for creating the FE models;
- Macroscopic approaches are concerned with the overall product data structuring and with the sharing and reuse of product data among applications.

Both microscopic and macroscopic approaches are required to address the issue of making the DMU content and structure match with the needs of large assemblies FEA.

This section is more dedicated on microscopic approaches. The first sub-section introduces the microscopic CAD-FEA integration methodologies for the preparation of FE models on both standalone components and large assemblies. The second sub-section spans relevant research works dealing with the issue of integrating functional and topological information of components and interfaces within DMU environments. A last topic, not really addressed in the literature concerns the DMU product structure adaptation, since analysts often need that the organisation of the components represented in the DMU matches with the structure of their simulation model.

The macroscopic approaches for product data structuring and knowledge capitalization for sharing and reusing product data among applications are addressed in Chapter 8 which is dedicated to the product data models that might support these transformations.

7.2.1 DMU shapes transformation

Substantial simplification of the design geometry is required to create a usable analysis model for the FEA. In order to get a simpler mesh and speed the computation, FE models are generated based on details removal, shape simplification and CAD models idealization.

To automate the creation of analysis models, the operations must use knowledge of the design to automatically create the analysis model. Armstrong, et al. use the idea of a priori knowledge and a posterior analysis of the results to make appropriate idealizations. Additional operations, such as medial-axis transform, dimensional reduction, and feature removal are used to create the analysis model [Armstrong, 1994, Armstrong et al., 1996]. Authors also describe the operations that allow analysts to suppress details and reduce the dimensionality of the part. Detail suppression is used to remove the geometric features that cause disturbances in the stress field. Finally, the idealization operations, as presented are automated by use of command files. These contributions did not address explicitly the relationship between detail removal and idealisation.

[Léon&Fine, 2005] describe how an appropriate geometric model and a set of geometric operators may significantly improve the efficiency of the FE model preparation phase. The geometric model is

associated to a set of operators enabling skin detail removal, topological changes, manifold changes (dimension reduction).

[Ferrandes et al., 2009] extend the approach associating these operators to mechanical hypotheses/criterion to bring an objective estimation of the model simplification and control the component shape changes. If a shape detail removed during the shape simplification process proves to be influent on the mechanical behaviour, it can be re-inserted on the simplified model, so readapting the initial simulation model.

In industrial CAE or CAD software, a set of geometric approaches are available to apply shape transformations to solids. Although automated operators exist, they are currently effective on simple configurations of standalone components. To process complex models, the user interactively modifies the object using shape transformation operators according to his/her appreciation priori appreciation of the simulation model created. There, a model preparation reduces to a global geometric operator without connection to criteria derived from simulation objectives and hypotheses.

More recently, researches concentrated on the identification of specific regions to automatically subdivide a complex shape before meshing. First, [Chong et al., 2004] propose operators to decompose solid models based on concavity shape properties before the mid-surface extraction to reduce dimensionally the model. [Robinson et al., 2011] propose to decompose thin/thick sections and produce a mixed-dimensional shell as simplified model. [Makem et al., 2012] propose shape metrics to analyse a part and identify automatically long, slender regions within a volume body. Finally, [Nolan et al., 2013] propose to automate the creation of mixed dimensional meshes based on the concept of **simulation intent**. The idea is to capture the explicit link between the simulation objectives and modelling intents at the beginning of the analysis process such as mesh dimensionality and type. Doing so, many otherwise manual processes can be automated. Using non-manifold modelling, authors propose to use automatically gleaned interface data that can be mapped from one dimensionality to another using “equivalence” (i.e. fine grain associations are captured between topology and features of the “base” solid model and derived reduced models). Thus, one mesh model and can lead to various simulation models through definition of new simulation intents. Changes to the base model are automatically propagated to the downstream simulation models through recalculation of the interface data and mesh equivalences.

These research works enforce the significance of CAD and FEM region decomposition to speed-up the FEA process. However, most of these decompositions are only available for specific configurations extracted from isolated components and essentially incorporate geometric criteria. These approaches still face difficulties to obtain consistent results on single mechanical components; similar approaches for large assembly models have not been demonstrated yet.

Few authors have studied the problem of assembly simulation preparation. Either the feature suppression method of [Gao et al., 2010] or the surface simplification of [Andújar et al., 2002] considers an assembly as a single solid and not as a component structure with functional junctions. To avoid the interactive generation of component interfaces, some CAE software are able to automatically detect interfaces into an assembly. However, the algorithms look for face pairs characterized under a global tolerance of geometric proximity to define contact areas and are not defining the non-manifold interface area. It appears also that component interfaces in DMUs are not restricted to contact areas [Shahwan et al., 2012]. [Clark et al., 2008] propose to detect these interfaces and create a non-manifold representation of the assembly with CUBIT software before meshing. [Boussuge et al., 2012] underline the importance of the interfaces between adjacent volumes to generate conformal assembly meshes. However, authors do not consider the relationship between interfaces and the simplification

and/or idealisation part processes. [Quadros et al., 2010] propose a framework to generate size functions controlling assembly meshes. Other authors like [Chouadria&Veron, 2006] identify a re-mesh contact interfaces in polyhedral assemblies. However, these methods are used directly on already designed mesh without establishing a link between CAD and CAE models and are restricted to contact interfaces.

Finally, [Hamri et al., 2008, Drieux et al., 2007] focus on the main characteristics of product views regarding the shape transformations that are needed to generate a suitable shape description and reference model for a specific simulation. They develop the product view interface concept based on mixed shape representation generated and provided for the specific objectives of the simulation task.

The above review shows that CAD-CAE integration is currently focused on standalone components; preparations of assembly models have not been addressed in depth under global simulation objectives. An assembly can be regarded as a set of components interacting with each other through interfaces. These interfaces contribute to mechanical functions of components or sub-assemblies [Kim et al., 2004]. An assembly simulation model derives from shape transformations interacting with these functions to produce a mechanical model containing a set of domains discretized into FEs connected together to form a discretized representation of a continuous medium [Boussuge et al., 2012]. Therefore, assembly simulation models, not only suppose the availability of geometric models of components, but they must also take into account the physical interfaces and behavioural interactions of the entire assembly as needed to reach simulation objectives. This suggests two requirements:

- the entire assembly must be considered when specifying shape transformations rather than reducing the preparation process to a sequence of individually prepared parts that are correctly located in 3D space [Boussuge et al., 2012].
- explicit functional and topological description of interfaces and interactions are required within the DMUs.

7.2.2 Explicit Functional and Topological description of Assemblies in DMUs

Unlike modelling a standalone component having no adjacent component, an assembly simulation model must be able to transmit displacements/stresses from one component to another. Therefore, the preparation of an assembly model compared to a standalone component implies a preparation process of interfaces connecting components together. According to [Boussuge et al., 2012], to obtain a continuous medium, the analyst must be able to monitor the stress distribution by adding or retrieving either kinematic constraints inside the assembly model or prescribing a non-interpenetration hypothesis between components by adding physical contacts. Thus, modelling hypotheses must be expressed by the analyst at each interface of the assembly [Boussuge et al., 2012]. To express the right modelling assumptions and requirements for their integrated FE models, integrators and/or analysts need to get the design intent of the assembly which is expressed more precisely by the functional and topological description of interfaces and interactions within the DMU.

[Kim et al., 2004] introduce a design formalism for collaborative assembly design to capture joining relations and spatial relationship implications between assembly parts. This modelling notation allows the joining relations to be described symbolically for computer interpretation, and the model can be used for inferring mathematical and physical implications. Based on this formalism, an assembly relation model and a generic assembly relationship diagram are generated to be shared / exchanged with co-designers.

The use of **oriented** or **directed graphs** to represent an assembly product model is a relatively common approach. [Zheng et al., 2006] introduce a theory of directed acyclic graph (DAG) and related concepts such as such as scene, entity, scene graph, linear scene graph, nonlinear scene graph.

A **Scene** is a role management and visualization space like a "stage" in a virtual prototyping system on which it plays all sorts of roles, consisting of the entity objects with geometrical elements to be displayed, the world coordinate system to position the entities and the lighting objects.
Entity is a graphic object to be displayed, consisting of geometrical elements and their status on the scene whether it is a product assembly or a part.

[Zheng et al., 2006] propose a modelling method for virtual prototyping based on DAG. In this method, the entities are managed with DAG by the scenes. Authors developed an application where the scene graph model, based on DAG and expressed in a neutral graphic format, describes geometric constraint relation as a node and interoperate with CAD systems to manage various "DMU assembly scenes" in a collaborative context.

In [Ballu et al., 2006, Falgarone&Chevassus, 2006], authors present research works performed at the Research Centre of EADS dealing with tolerancing. Authors propose a systematic approach for representing and handling complex assemblies with thousands of parts with many functional requirements. The proposed method integrates GASAP, an approach for modelling parts, assembly and tolerance specifications in a CAD system. The method is supported by GAIA, a new software tool built on assembly-nested graphs. The GAIA software enables to describe functionally the product, specify interfaces and constraints (functional and dimensional) and propagate cascading requirements between the different levels of graphs (see Figure 51).

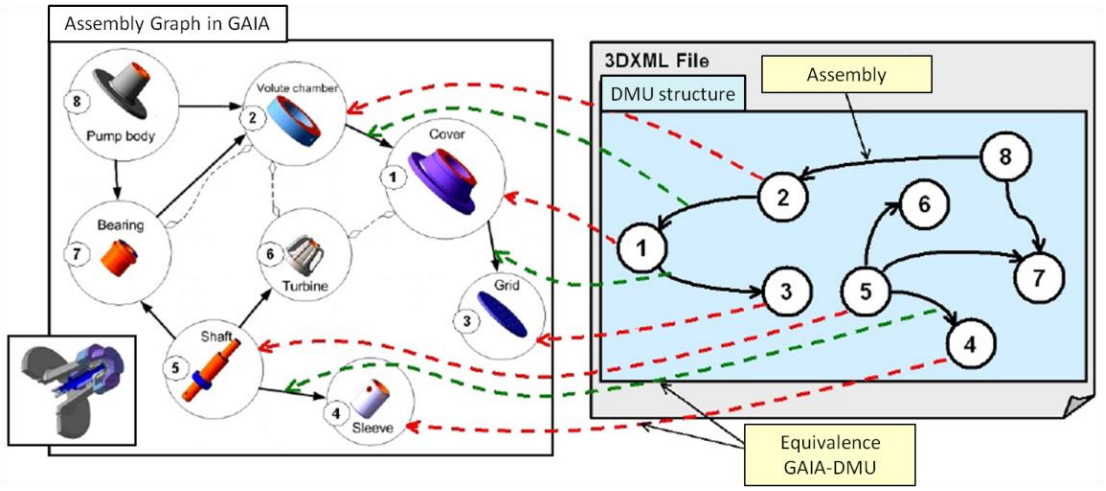


Figure 51: Correspondence between an assembly graph in GAIA and the equivalent DMU structure in a 3DXML file from [Drieux, 2006]

In [Iacob et al., 2008] a process and a framework for contact identification is presented. The proposed method provides a smarter way to manage collisions, using the contacts information. The process is automated by a contact identification operators (identifying the common area between components) combined with the topological description of partitions of the geometric model. Then, the assembly is analyzed and all the information about contacts is stored in a data structure. The approach only addresses specific types of contacts and is only applied for better collision detection and kinematic constraints processing in haptic devices simulations.

In [Demoly, 2010, Demoly et al., 2010a, Demoly et al., 2011b, Demoly et al., 2011d], an integrated framework entitled Proactive ASsembly-Oriented DEsign (PASODE) is introduced. This framework,

based on a multiple views data model called MUltiple Views Assembly Oriented (MUVOA) (see section 8.1.2.4), enables to automate the definition of assembly sequences as well as the definition of a skeleton-based assembly context in the preliminary product design process. To achieve these automations, authors introduce the concept of "**Bill-of-Relations**" and a set of successive procedures are required to capture four kinds of assembly components relationships that are captured and exchanged via XML files: assembly-components decomposition relations, physical contact relation between two components, kinematic relations and technological relation (defining the mating relation between two components in contact). The proposed MUVOA model and PASODE framework have been implemented in a prototype application called PEGASUS at the interface of PDM systems, MPM (Manufacturing Process Management) and CAD systems. Starting from a product structure imported from the PDM system, a liaison graph describing contact relations and assembly pairs between product components is defined in PEGASUS. This latter provides the so-called PDM-oriented 'Bill of Relations' describing composition, interface, and representation links between product components in the PDM system. This Bill of Relations (BOR) served as input for automating the definition of assembly context in preliminary design CAD models. A CATScript use the xml-based bill of relations to generate the product structure in CATIA including parts and sub-assembly CAD documents. Then, the PEGASUS CAD Assistant makes use of this data structure to assign, through each structure level, a parameterised assembly skeleton. Figure 52 shows the import feature of "Bill of relations", and a display of graphs defined in the PEGASUS application. It also shows the PEGASUS CAD Assistant helping in the definition of kinematic/technological relations between product components, hence permitting to automatically build the skeleton entities in the CATIA v5 environment.

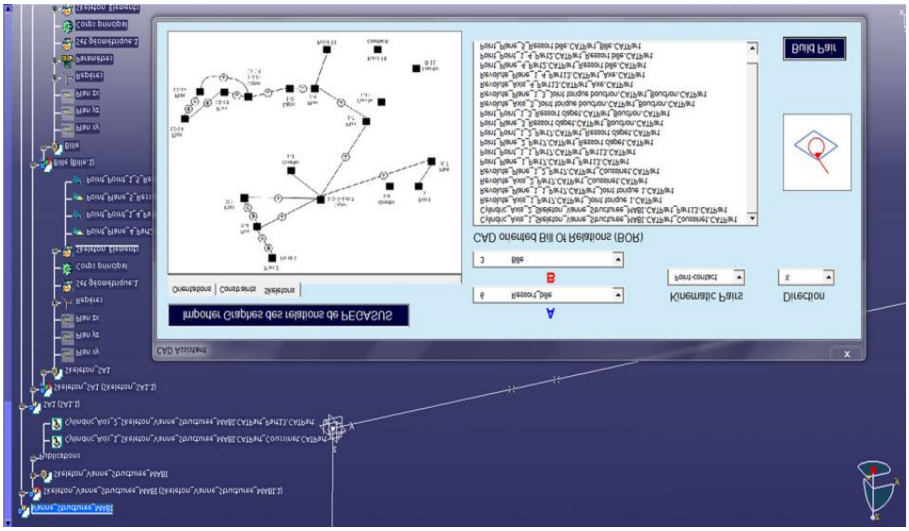


Figure 52: Skeleton entities definition via PEGASUS CAD Assistant within CATIA v5 from [Demoly et al., 2011a]

The extraction of functional data from a DMU through a bottom-up approach as the one conducted by [Shahwan et al., 2012] demonstrates its efficiency in characterizing functional interfaces in a mechanical assembly. The authors identify the functional designation of components through a combination of their geometric interactions with a qualitative mechanical reasoning process. This approach shows that the geometric interactions between components in a DMU are not only contacts and clearances but can be interferences, which leads to the concept of **Conventional Interfaces**. A conventional interface is initially defined by a geometric interaction that can be a contact or interference between two components [Shahwan et al., 2012]. This bottom-up process starts with the generation of a Conventional Interfaces Graph (which is an oriented graph) with components as nodes, and conventional

interfaces as arcs (see Figure 53). Conventional interfaces are then populated with **Functional Interpretations** according to their geometric properties; producing potentially many combinations (see Figure 54).

Figure 53: Conventional Interfaces Graph of a simple cap-screw model from [Shahwan et al., 2011]

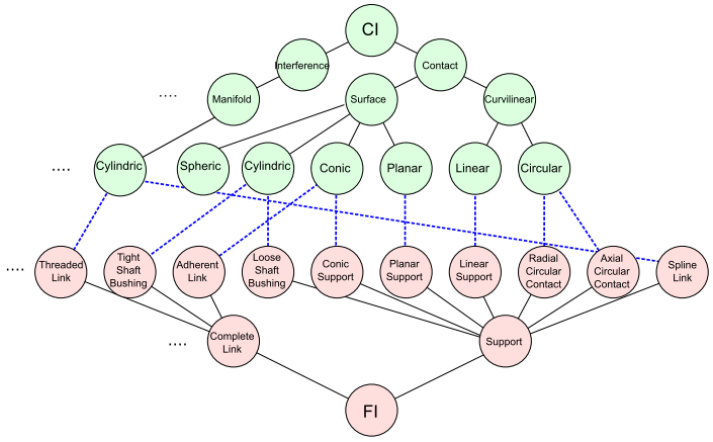


Figure 54: Conventional interfaces and Functional Interpretations combinations from [Shahwan et al., 2012]

Authors also introduce the concept of DMU state which describes a physical and qualitative behaviour of a DMU through equilibrium equations. A behaviour law is applied to each component of the DMU where each interface is assigned a possible functional interpretation. States and design rules are introduced to express the elementary behaviour of some DMU components (e.g. a spring relaxed or not) through a qualitative reasoning process. This reasoning process, based on domain knowledge rules, checks the validity of certain hypotheses considered to hold true during a specific stage of the design process. This verification against reference states reduces the number of Functional Interpretations per Conventional Interfaces. Domain knowledge rules are then applied to group semantics of components interfaces into one functional designation per component to connect together geometric entities of its boundary with its function [Shahwan et al., 2013]. The objective of the approach is to provide the basis to automate the shape transformations of components and interfaces during an assembly preparation process.

Finally, based on the works done by [Hamri et al., 2008, Foucault&Léon, 2010, Foucault et al., 2011, Shahwan et al., 2011a, Shahwan et al., 2011b, Shahwan et al., 2012, Shahwan et al., 2013], [Boussuge et al., 2012] analyze the content of a DMU and explain why information about interfaces between components is missing in DMUs and how to derive the shape idealization of industrial assembly models, resulting in categories of DMU transformations. Authors propose a methodology for automating the preparation of assembly FE models. The methodology consists in using the identification of functional features of the assembly through the interfaces between components to locate groups of components related to similar assembly functions [Shahwan et al., 2013] and set a connection with the simulation objectives. The simulation objectives are expressed through user-defined hypotheses on shape transformations. According to authors, the multiple idealizations and interfaces between

sub-domains can generate repetitive patterns that can be re-used in order to speed-up the FEA preparation processes. This preparation process is given on Figure 55.

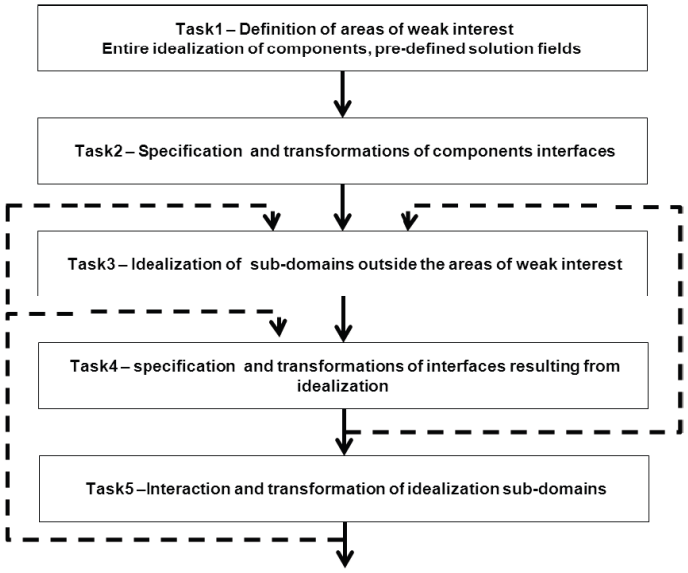


Figure 55: Structure of an assembly simulation preparation process [Boussuge et al., 2012]

7.2.3 Simulation-driven DMU structural transformation

As mentioned in Chapter 4, when dealing with large DMUs (referencing thousands of CAD models), very few analyses require the whole product DMU. Most of them require a subset of it. Moreover, analysts often need that the organisation of the components represented in the DMU matches with the structure of their simulation model. It is obviously not the case in "As-Is" DMUs.

To make the DMU the product definition referential [Drieux, 2006] underlines the importance for PDM system to support different organizations of the DMU to offer to different actors a DMU structuring corresponding to their point of view and business needs.

Structural transformations refer to the operations related to the reorganization of groups of components, independent from their shapes, that is to say, the modification of their hierarchical interrelationships within a product structure. These types of DMU transformations leading to "DMU scenes" is not really addressed, or at least not explicitly, in the literature. The tools to achieve this are also rare whereas the required technology already exist. According to [Drieux, 2006] there are several reasons explaining this observation:

- Formats for describing a structure as a tree or graph are not standardized and uncommon, which does not facilitate exchanges between software. However formats as 3D-XML or STEP can capture the DMU structure or the PDM tree for further processing. The lack of specific formats for the structural description of DMU assemblies makes this description restricted because of the export capacity of the geometric format used by the CAD software.
- Therefore, in many cases, the transition to a format suitable for downstream application results in a loss of all or part of the structural information. In some cases the receiving system can import a structure of component groups as a tree. If the tree can be imported within the system, the reorganization of its structure, if permitted by the system, is often the responsibility of the user and hence manual.
- The organization of components for the visualisation of DMU scenes is generally managed by the CAD software in which the DMU is visualised and where the scene is displayed. These tools

already have their own specific data structure optimized for their own operation, which justifies that they do not necessarily need to use a possible pre-existing standardized data structure. For instance, collision detection tools use subdivision and decomposition algorithms of the space containing the scene to optimize the number of intersection tests between pairs of objects. 3D rendering engines also use these kinds of algorithms to reorganize themselves objects according to distance or materials criteria.

Existing approaches to semi-automatically reorganize a DMU are often applied in the field of “Virtual reality” and “Design For Assembly” mainly to perform assembly/disassembly trajectories simulations within CAD environments. [Jayaram et al., 2004] present a set of developed tools enabling a semi-automatic reorganization of a product from a tree view "As designed" extracted from a CAD software to a view "As planned" corresponding to the actual simulated assembly sequence. [Graf et al., 2002] propose a mechanism to perform a mapping between the CAD tree and the structure of virtual model as a tree scene taking into account different product configurations, the multiple instances of CAD parts and using external references for the geometric elements.

[Drieux, 2006, Drieux et al., 2007] and [Ballu et al., 2006, Falgarone&Chevassus, 2006] underline that an assembly described by directed graph, specifying relations between parts in an assembly, enable to consider reorganizations of different assembly sequences. We also believe that such an approach can enhance automated or semi-automated generation of simulation-driven DMU sequences/structures/scenes.

[Kibamba, 2011] also underlines the important requirement of providing new adapted product structures fulfilling specific simulation requirements. The author proposes a methodology consisting in first enriching product structures with fluid elements not traditionally present in DMUs (required to generate CFD models). Then the structure is enriched with components interfaces elements providing the functional and kinematic definition of mechanical interfaces but also includes fluid-structure interfaces. Finally the methodology also consists in using the interface elements definition to generate a directed graph of the corresponding assembly. Based on this directed graph, the author proposes a procedure to reorganize components according to the kinematic definition of their interfaces (grouping embedded components).

7.3 From DMU to BMU and integration with PDM systems

Design and structural behaviour simulation are not regarded as two independent disciplines any more. [Eckard, 2000] showed that the early integration of structural simulation in a design process could improve a PDP leading to a shorter time-to-market. To help analysts, [Troussier, 1999] [Peak et al., 1999] and then [Bellenger et al., 2008] formalized simulation objectives and hypotheses applied to a design model when setting up simulations to capitalize and reuse them in future model preparations. The approach of adapting PDM systems to numerical simulation activities was taken over by [Klaas&Shepard, 2001] and then [Shephard et al., 2004]. In their approach an emphasis is placed on the technical components that must be added to existing CAD and CAE tools to enable the application of simulation-based design. The authors propose a called SEED environment, based on of the CAD/FEA integration principles set by [Arabshahi et al., 1993]. As shown on Figure 56, SEED components include a simulation model manager, simulation data manager, adaptive control tools and simulation model generators.

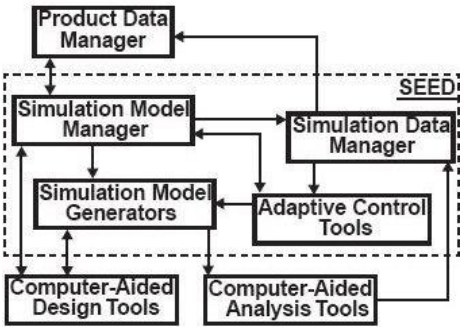


Figure 56: SEED environment architecture from [Shephard et al., 2004]

The BMU (see Figure 57) is the equivalent of the DMU for simulation data and processes. Beyond the geometry, which is represented in the DMU, the so-called BMU shall logically link all data and models that are required to simulate the physical behaviour and properties of a single component or an assembly of components [Riel, 2005]. BMU shall use all the DMU-data it needs for model calculation, and all relevant BMU calculation results shall be made available and physically accessed via the DMU.

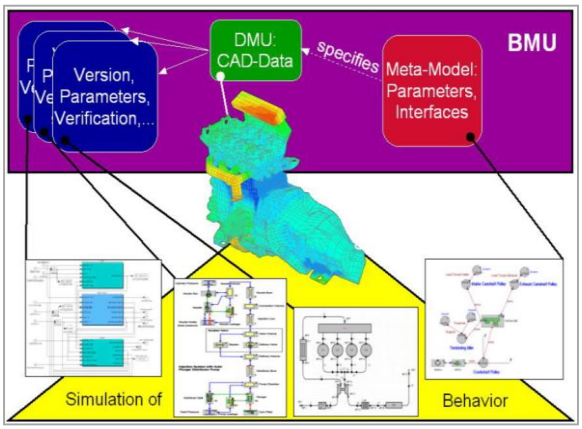


Figure 57: Meta-modelling Concept of the BMU from [Riel, 2005]

Within this relationship, the DMU shall serve as the key link between the BMU and the PDM-system. These requirements demand a concept to handle the structural information that is necessary to determine the relationships among components and to map DMU and BMU content and structures [Riel, 2005].

In [Nguyen Van, 2006] the author proposes a centralised architecture to ensure multi-partnership collaborative design and “multi-view engineering” by providing a common referential for data semantic in order to ease the migration of data between a PDM system and a SDM system. Based on STEP standards, a prototype has been developed during the VIVACE project: the Engineering Data Management (EDM) framework. The aims of the EDM framework was “to manage engineering data in a broader perspective than the current aeronautics engineering activities bounded to the static DMU view by encompassing requirements domain, product domain as well as simulation data” [Tabaste, 2005]. An emphasis was placed on all the characteristics inherent to a common framework link between design and simulation domains. In this project, partners have introduced the concept of “heavy simulation interface” which is a CAD-CAE integration approach requiring several capabilities [Tabaste, 2005]:

- The management of the large data sets between the activities of SDM and PDM. This part concerns the definition of common repositories in which information is shared by the different activities and partners;
- The management of the large data sets between the activities of simulation and design. This part defines the global relation and coherence between design data and associated models and simulation data and linked models.
- The management of the functionalities required for simulation activities. For example to enable the relation between a simulation model and a design model for the assembly interfaces in order to keep a global coherence in the study of the product.
- The management of the hardware architecture and infrastructure in order to have a physical view on the interface definition.

The European CRESENDO project is the following of the VIVACE project. The project aims at delivering the modelling and simulation backbone of the aeronautical extended enterprise: the **Behavioural Digital Aircraft** (BDA). The BDA concept represents the BMU of the aeronautics extended enterprise and might consist in a collaborative data exchange/sharing platform for simulation processes and models throughout the development life cycle at aircraft level and in the entire supply chain extending the concept to the "Mastered Behavioural Digital Aircraft" (MBDA). In CRESCENDO, the MBDA is considered to comprise:

- The "Behavioural Digital Aircraft" (BDA), as a federated and orchestrated suite of enabling capabilities for models data and information management (the Model Store), together with simulation process management (the Simulation Factory), including modelling and simulation quality management methods and procedures (the Quality Laboratory), and supported by information and knowledge sharing to enable cross-enterprise decision-making (the Enterprise Collaboration).
- The complete range of models and simulations needed by multiple overall aircraft design views that describe the behavioural, functional and operational aspects of the whole aircraft and constituent systems (e.g. engines, avionics, fuel systems), sub-systems, and components.

A schematic representation of the expected BDA architecture and the CRESCENDO use cases that have permitted to define it is provided in Figure 58 below.

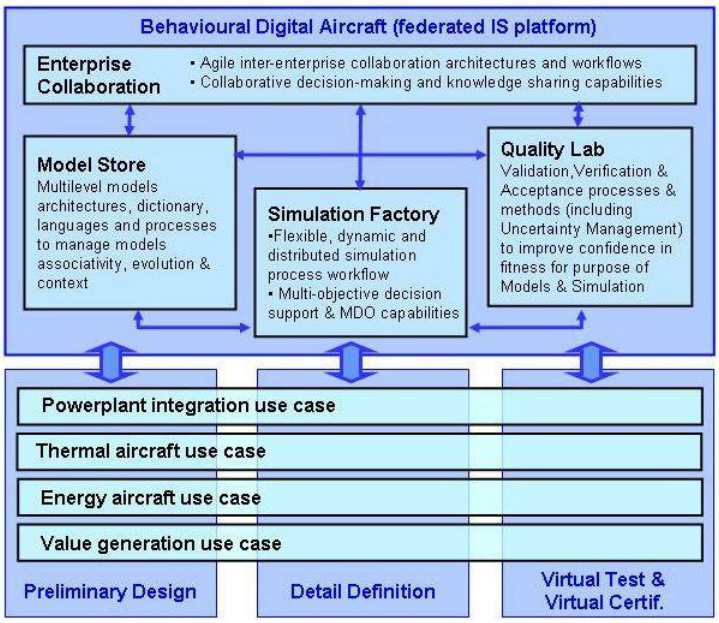


Figure 58: Behavioural Digital Aircraft Enabling Capabilities and Use Cases

Another French consortium research project called ADN ended in 2010. This project aimed at capitalizing, reusing and managing design knowledge to ensure capitalization, traceability, consistency and reuse of design and simulation parameters, business rules in the design process and throughout the product life cycle. In [Badin, 2011, Badin et al., 2011] authors proposed a method of knowledge management used in several interacting activities within a design process. There, analysts and designers collaborate and exchange design information. However, the authors assume that relationships between dimensional parameters of CAD and simulation models of components are available, which does not currently exist. Additionally, they refer to configurations where the shapes of components are identical in the design and simulation contexts.

Finally a French research project called ROMMA (RObust Mechanical Models for Assemblies) and financed by the French National Research Agency (ANR) has been launched in 2010 and will end in 2014. The aim of this project is to remove a number of scientific locks on the modelling and simulation of the behaviour of assemblies of mechanical structures applied to industrial cases with a large number of fasteners. Based on the statement that for such situations much useful information for the simulation are absent from the initial 3D geometry (CAD), the project focuses on the development of enrichment strategies of geometric model in order to automatically create simplified simulation models. It also covers the construction of automatic 3D local calculation model and their use in the framework of re-analyses around a few local fasteners.

7.4 Conclusions

Simulation-Based Design, as a part of MBSE, relies on the use of consistent design models representing the system or sub-system to analyse. While dealing with large assembly simulation models, assembly information such as geometric interfaces must be specified and captured in the design models. MBSE object-oriented modelling techniques can help focusing on these interface specifications but also providing and linking consistent multiple views of the product.

In mechanical product design, one of the key federating environments to exchange/share product definition data is the DMU. Indeed, while enhancing 3D and 2D simulations – that use 3D and 2D CAD models as input – in a collaborative and distributed design process, the DMU appears as a simulation data inputs referential permitting to speed-up the simulation preparation process.

However this chapter has underlined the DMU transformations required to provide adapted DMUs that can be used as direct input for large assembly FEA. These transformations must be consistent with the simulation objectives. Three types of transformation have been identified:

- DMU shapes transformation: current researches on this topic mainly focus on standalone components; preparations of assembly models have not been addressed in depth under global simulation objectives.
- DMU enrichment in terms of functional and topological components' interfaces: the most relevant work on this topic use directed graphs to represent an assembly product model enriching the DMU content with interface information.
- DMU structural transformations: These types of DMU transformations leading to "DMU scenes" is not really addressed, or at least not explicitly, in the literature.

During the last decade significant research efforts have been made and several R&D consortium projects (gathering industrial and academic partners but also software editors) have been launched in order to define how to provide and implement the BMU as well as its links with the DMU and the PDM systems. The aim of these projects is to be able to:

- Integrate both CAD and CAE data in a common engineering data management framework.
- Formalise simulation objectives and ensuring traceability between CAD and CAE data to enhance re-use of CAD and CAE models for downstream simulations
- Provide capabilities to adapt CAD data inputs regarding simulation objectives and speed-up the input data acquisition as well as automating the mesh generation process.

However efforts remain necessary to provide the essential missing link between the PDM-based DMU and the SDM-based BMU.

A barrier preventing to achieve these objectives is that CAD and PDM systems used as demonstration platforms are supported by data models that are not currently adapted to make explicit links between CAD and CAE data and to provide consistent multiple views of the product regarding the simulation objectives. Indeed, when using an integrated system model, the modelling notation must explicitly allow for information to be shared in different views [Shah et al., 2009]. Implementing such a formalism in **PDM and PLM systems requires an evolution towards better standardization, data consistency, and concentration on a few robust sources, and that the MBSE approach must be part of this new PLM strategy [Alemanni et al., 2011]**. Therefore, there is a crucial need to improve the product data models that support both PDM and SDM systems and to implement MBSE concepts in these digital collaborative platforms.

Chapter 8: Impact on Product Data Models and PLM systems

This chapter aims at identifying the existing product data models enabling to manage and represent product design information according to various discipline-specific aspects of a design artefact. It includes a literature review of existing standardised and non-standardised product data models supporting:

- Generic product data management capabilities: such as product identification and classification, product components and other design artefacts definition, components' physical properties and shapes, product structure and configuration management capabilities;
- Explicit description of assemblies structures and components' interfaces: this concerns product data models encompassing specific objects and methods to capture the functional and topological description of components interfaces;
- The continuity and traceability of CAD and CAE data at different levels of abstraction and the re-use of design artefacts, models and knowledge in the appropriate context;
- The exchange and consistency of information across multiple multi-domain and multi-level views of the system.
- The propagation of engineering changes across these multiple product views.

NB: The STEP-based data models presented in the following literature review, and which are originally defined in EXPRESS and represented in EXPRESS-G, are represented here in UML class diagrams. This effort has been done for a better understanding and to avoid ambiguity and potential varying interpretation of this complex standard and related concepts and definitions.

8.1 Multi-aspects product data and meta-data models

Managing, accessing and integrating information from multiple scientific data sources is a major challenge for product design [Feng et al., 2009] and is tightly coupled with the existence and the evolution of **product data and meta-data standards** [Krause&Kaufmann, 2007]. Such standards for detailed geometry-related product data are important for consistent interpretation of product geometry specification and verification, and for interoperability among engineering tools such as CAD, CAM and CAE systems. But to reach out to other engineering needs, to make heterogeneous PDM systems interoperable and to make these product data contextualized and interpretable by the data consumer, standardized product meta-data models are required. **Product meta-data** are "data describing the data" or "data about the data" and in our scope, this covers such information as author, approver, version, change history, configuration data, etc. as well as "aggregate data" such as part number, product assembly structure, etc. Traditionally, PDM systems defined and managed the product meta-data, leaving the bulky, detailed geometric and other data to the CAD, CAM, and CAE systems [Srinivasan, 2011].

This section first introduces existing standardized and non-standardized generic product meta-data models proposed in the literature. Secondly a review of existing product data models supporting the explicit functional and topological description of assemblies is given. Product data models supporting tight integration and traceability between CAD and CAE data as well as re-use methodologies are then identified. Finally the section ends by identifying the current product data models and gaps related to the exchange of consistent information across multiple multi-domain and multi-level views of the system and to the propagation of engineering changes across these multiple product views.

8.1.1 Generic product data management capabilities

8.1.1.1 Introduction to ISO 10303 - STandard for the Exchange of Product model data

ISO 10303, also known as **STEP** (STandard for the Exchange of Product model data), is an international standard for the representation and exchange of product model data. The objective is to provide a mechanism that is able to describe product data throughout the lifecycle of a product, independent from any commercial system. STEP was primarily developed with the purpose of developing a vendor-independent and neutral exchange format of CAD data describing both product structure and geometric information [Kemmerer, 1999]. However, STEP's scope has evolved into a much broader scope than that of other existing CAD data exchange standards, notably the Initial Graphics Exchange Specification (IGES), a US standard that has been in use for more than 20 years. Whereas IGES was developed primarily for the exchange of pure geometric data between CAD systems, STEP is designed to handle a much wider range of product related data covering the entire life cycle of a product [Pratt, 2005]. This range is continually expanding as new parts of the standard are issued. However the majority of STEP translator implementations concerns only CAD applications and are only used to exchange/share structural and geometric CAD data either between heterogeneous CAD systems [Gerbino, 2003], either between heterogeneous PDM systems or between CAD and PDM systems [Oh et al., 2001].

The entities to be captured and exchanged using STEP, and their relationships, are defined in schemas written in an object-oriented information modelling language called **EXPRESS** (Schenk and Wilson, 1994). The syntax and related information of EXPRESS are described in ISO 10303 — Part 11 [ISO, 1994b]. **EXPRESS-G** is a subset of the EXPRESS language supporting the graphical notations of schema, entity, type and their relationship concepts.

STEP defines a number of data models for various aspects of product data. The ISO10303 encompasses six main categories of standards called **Parts**. Individual parts are referred to as ISO 10303-xxxx, where xxxx is the part number, and each is a standard in its own right, though it is interdependent on other parts and consequently a component of a larger whole. The Parts are organized into seven groups as follows [ISO, 1994a]:

- Description methods — Parts 11—19;
- Implementation methods — Parts 21—29;
- Conformance testing methodologies and framework —Parts 31—39;
- Integrated generic resources — Parts 41—99;
- Integrated application resources — Parts 101—199;
- Application protocols — Parts 201—1199;
- Abstract test suites — Parts 1201—2199.

The STEP information models are built with a three-layer structure, i.e., the physical layer, the logical layer, and the application layer. The principal product representation entities for all phases of product life cycle are defined in the logical layer. Such entities are classified by their properties into several specific parts called **Integrated Resources**. The EXPRESS language enables to classify and construct Integrated Resources by their data entities, attributes, rules, relationships, functions and constraints [Peng&Trappey, 1998]. The core integrated resources of STEP mainly used in existing application protocols are:

- Part 41 - Fundamentals of product description and support
- Part 43 - Representation structures
- Part 42 - Geometric and topological representation
- Part 44 - Product structure configuration

The application layer provides the **Application Protocols** (AP) to support various types of applications (e.g. automotive and aerospace design, ship building, printed circuit board, etc.). Each AP consists of the relevant data entities, relationship, and some constraints for a specific application domain. Some entities exist as the integrated resource models, defined in the logical layer [Peng&Trappey, 1998]. The first section of an AP describes what is in and out of scope for data exchange. Then each STEP AP encompasses:

- an **Application Activity Model** (AAM) which describes the intended context and the process that the AP enables (written in the IDEF0 modelling language);
- an **Application Reference Model** (ARM) which describes the application view of the product data providing a documented information (data) model of all of the information requirements of the AP;
- an **Application Interpreted Model** (AIM) which is an information model that specifies the normative part of the standard. An AIM is a specialized subset of the Integrated Resources that is the result of the mapping of the ARM requirements information model to the STEP integrated resources information models.

The goal of this structure is to avoid duplication of work and to enable APs to “speak” more or less the same language. To support the conformance testing of STEP implementations so-called **Conformance Classes** are defined for each AP. These Conformance Classes (CC’s) are subsets of an AIM with additional testing and instantiation procedures [Gielingh, 2008] so that the standard can be implemented "meaningfully" within that application domain without having to implement all aspects of the AP. Each CC consists of a group of one or more **Units of Functionality** (UoF). Implementation of selected conformance classes can be seen in those AP's that have been commercially implemented to date (i.e. AP203 and AP214). These CCs specify subsets of the total AP content that must be completely implemented by STEP translators if they are to claim conformance with the standard. For instance and as shown in Table 1, in the AP214, CC1 and CC2 cover the part and assembly geometry data. CC 6 and CC 8 were created to cover product meta-data handled by PDM systems that can treat geometric model data as files.

AP214	Class	Description	Content
Data (Engineering objects)	CC1	Component design with 3D shape representation.	Covers 3D geometry of single parts, including wire-frame, surface, and solid models.
	CC2	Assembly design with 3D shape representation.	Covers 3D geometry of assemblies of parts, including the assembly and model structure.
Meta-data (Business objects)	CC6	Product data management (PDM) without shape representation.	Covers PDM systems that manage geometric models as files. It also covers administrative data of parts, assemblies, documents, and models.
	CC8	Configuration controlled design without shape representation.	Covers CC 6, with additional requirements for product configuration control.

Table 1: Standardized product data and meta-data managed in ISO STEP AP214 conformance classes [Srinivasan, 2011]

STEP AP214 has several other CCs (20 in total), but these four are the ones that are currently supported by CAD and PDM vendors. It is not enough to indicate that an application has a STEP or an APxxx translator. The most important is to know what conformance classes of the AP have been implemented and to understand the coverage of those conformance classes. For instance, the AP214 conformance classes 1 and 2 represent a subset of this AP that is roughly equivalent to AP 203 and most vendors who claim to have an AP214 translator have only implemented cc1 and/or cc2 that are

essentially identical to AP203 geometry/topology related CCs with a somewhat different set of configuration management data.

More than 40 application protocols have emerged from the ISO 10303. The main areas addressed by STEP are currently mechanical design, manufacturing, electronics, shipbuilding, architecture and civil engineering. In the field of mechanical product design, the key APs enable to exchange data related to technical drawing (AP201, AP202), 3D modelling with configuration management (AP203), structural finite element analysis data (AP209), automotive design (AP214), aerodynamic calculation (AP237). The STEP protocols AP203 and AP214 are the mostly used ones in the domain of 3D CAD data exchange.

8.1.1.2 Overview of STEP Application Protocols

ISO10303-203 or STEP AP203 ("Configuration controlled 3D designs of mechanical Parts and Assemblies") is mainly used in the aerospace and defence industries by builders of aeroplanes and suppliers of engines [Gielingh, 2008]. It is also used by a few other companies and governmental bodies [PDES, 2006]. AP203 focuses on the design of manufactured product. Therefore to support their design and their configuration management, AP203 permit to describe:

- The global engineering context: the people and their roles, companies, dates, and the product / supplier / customer relationships, authorizations monitoring, data confidentiality, the measure units employed;
- The Product: its identification, its classification, some of its related data (drawing, contract, shape, ...), its reference if it is an external product, its structure, assembled parts and their positioned shapes, its evolution and history;
- The 3D geometric representation of the product: bounded wireframe models and surface models, wire-frame models with topology, manifold surface models with topology, faceted and non-faceted boundary representations.

The first version of AP203 did not take into account many data or features used by current commercial CAD systems and the models exchanged thanks to application protocols are frozen and most of the time not re-used. STEP AP203-based CAD models geometries are considered "dead" since it is not possible to resize or change function parameters. This is to close these gaps that the proposed definition of AP203 Edition 2 has emerged [ISO, 2005].

ISO10303-214 or STEP AP214 ("Core Data for Automotive Mechanical Design Process"), was originally emerged as an extension of AP203 but specifically developed for the automotive industry and its specific business needs. Its scope was wider including the same UoF than AP203's CCs plus new capabilities such as the management of raw materials data, material properties and simulation data (for the description of kinematic structures), process plan information (to manage the relationships among parts and the tools used to manufacture them), standard parts, tolerance data, features, numerical control (NC) and engineering change management. However the really addressed scope of AP214 in mechanical CAD is roughly equivalent to AP203, overall after the publication of the 2nd edition of AP203 (in which PDM modules have been harmonized with AP214). AP214 is applied by the automotive industry, especially by European car manufacturers. The uptake by US car manufacturers is minimal [Tassey et al., 1999, Gallaher et al., 2002, ISO-SCRA, 2006, Gielingh, 2008].

These two AP's cover most of the current commercial use of STEP. Although AP203 and AP214 are still mainly used to exchange CAD data describing product structure and geometric information be-

tween heterogeneous CAD systems, its usage can be and must be extended to the area of PDM systems. The automotive sector is now using AP203 and AP214 more frequently for the exchange of configuration management data [Gielingh, 2008].

The requirements to share geometric shape and analysis information in a large-scale system have been addressed by an emerging STEP standard: the **ISO10300-209**. **AP209** ("Composite and Metallic Structural Analysis and Related Design"). This AP specifies computer-interpretable composite and metallic structural product definition including their shape, their associated finite element analysis (FEA) model and analysis results as well as the material properties. The scope of AP209 is the product definitions of the analysis and design disciplines. The analysis discipline of AP209 primarily focuses on FEA models and analysis controls, results and reports. The design discipline of AP209 is concerned with shape representation of components and assemblies. AP209 provides an important mechanism for sharing information between analysis design models and a standards-based solution to iterative design-analysis integration problems and will be more specifically addressed in 8.1.3.

Previous relevant standards only address very specific areas of the overall product life-cycle. **ISO 10303-239** or **AP239** ("Product Life Cycle Support") (PLCS) is currently the only international standard available that intents to cover the entire product life cycle spectrum [Sudarsan et al., 2008]. Within the AP239 perspective, information need only be acquired once in the product life cycle, but may be used many times. Some of the key areas addressed by AP239 are:

- Product Description: the definition of product requirements and configurations, including relationships between parts and assemblies, in multiple evolving product structures (as-designed, as-built and as-maintained);
- Work Management: the request, definition, justification, approval, scheduling and feedback capture for product life cycle activities and their related resources
- Property, State and Behaviour: the representation of feedback on product properties, operating states, behaviour and usage
- Support Solution and Environment: the definition of the support required for a given set of products in a specified environment, and of support opportunity, facilities, personnel and organisations;
- Risk assessment and risk management: the representation of risk related data associated with the product life cycle.

Appendix VI provides the PLCS Concept Model which is a high level model of the main concepts used in ISO 10303-239. Because the information model defined by ISO 10303-239 (PLCS) has a scope that is wider than most applications, it is unlikely that any single software application will be able to declare compliance to the whole of PLCS. It is would also be difficult to contract for data to be provided according to the whole of ISO 10303-239 as the scope is so large. The **DEXs** (Data EXchange specifications) address this problem by providing a way of narrowing down the scope of the information model to be used in any given exchange. The PLCS DEX architecture is shown in Figure 59. There are a number of data structure patterns of the PLCS model that will be common to many DEXs. Rather than each DEX replicating the detailed specification of these data structures, "**Templates**" are defined for common elements of the model and are reused across different DEXs. Templates are defined within "**Capabilities**" (a description of how EXPRESS entities are used and related to represent a given concept and what *Reference Data* should be used) which, collectively, provide a complete usage guide for the PLCS model. Additional semantics may be represented by extending the entities of the generic PLCS information model through classification with so called "**Reference Data**". This provides a mechanism for

adapting the model to the semantics of more specialized domains [Eurostep, 2013] and hence for extending or tailoring it through the use of Reference Data Libraries [Pratt, 2005].

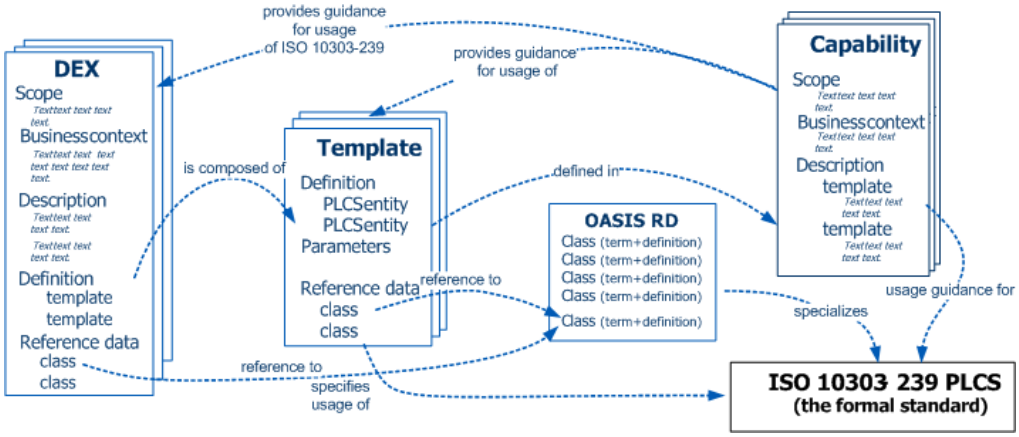


Figure 59: PLCS DEX architecture - DEXs, Capabilities, Templates, and Reference Data.

In common with all STEP APs, AP239 distinguishes between the semantics of the data and its manner of representation. This standard propose, based on the Gero's FBS framework [Gero&Kannengiesser, 2004] and system engineering concepts, to manage multiple evolving product structures by defining different types of product breakdowns among which functional, system and physical breakdowns.

In that sense, there is some overlap of AP239 with the capabilities of another developed STEP standard: **ISO10303-233** or **AP233** ("Systems engineering data representation"). The AP233 has its inception in the SEDRES project and resulted data models [Johnson et al., 1999, Johnson, 2000, Müller&Heimannsfeld, 2000] which objectives was to produce a workable "Systems Engineering data exchange standard", to progress with this standard in the ISO forum and to provide a set of prototype of data exchange tool interfaces. Originally AP233 UoFs were System architecture, Requirements, Functional design, Behavioural design, Data Types, Physical design / architecture, Properties, Graphics and Configuration Management. Today these UoFs have been reorganised and refined even if the scope of the standard remains the same. [Herzog, 2004] has classified these UoF according to 5 functional groups structuring the information model:

- System architecture – representing the building blocks for covering all information valid for a system, partial view of a system or system interfaces.
- Specification elements - defining the basic building blocks enabling to cover common specification techniques, including requirements, functional and physical architectures as well as verification and validation data;
- Requirement and functional allocation - defining the mechanisms for tracing requirements to functions (including behaviour), as well as physical architecture elements and functional architecture elements to physical architecture elements.
- Engineering process - covering the building blocks for activities in the engineering process, and associating specification information to related activities.
- Support information - representing the building blocks for representing supplemental systems engineering information. This large group encompasses configuration management information, visual layout information as well as mechanisms for referencing external documents, administrative information, data types and properties.

Figure 60 shows a UML conceptual view of the AP233 system model as defined in [Müller&Heimannsfeld, 2000, Düsing, 2000, Herzog, 2004].

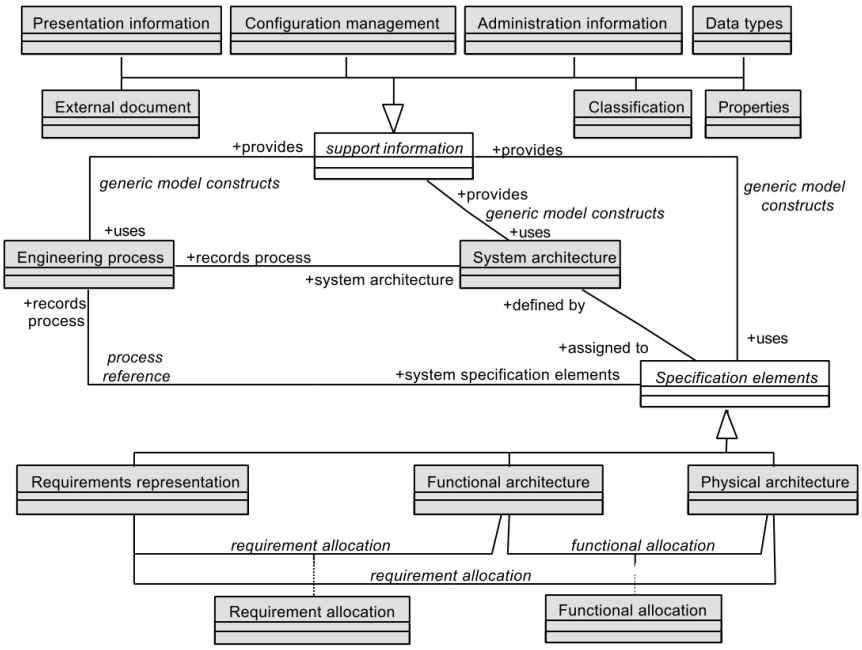


Figure 60: Conceptual view of AP233 from [Herzog, 2004]

The extensions to the original geometry oriented parts of the STEP standard as well as the concurrent development of APs such as AP233 and AP239 using common UoFs illustrate the relevance of the STEP development modular approach. In order to cover many aspects of the product life-cycle and to provide generic standards that can be used in various application domains, the trend is to develop STEP parts defined as intersection of functionalities present in a set of different APs.

The **STEP PDM Schema,** developed and maintained by two prominent standards consortia (PDES Inc and ProSTEP iViP), contains some of the most important standardised product meta-data models that came out of the ISO STEP efforts. The STEP PDM Schema is a reference information model for the exchange of a central, common subset of the data being managed within a PDM system. It represents the intersection of requirements and data structures from a range of STEP Application Protocols as shown on Figure 61. The PDM Schema covers 15 units of functionality (UoF) such as Part Identification, Part Classification, Part Structure and Relationships, Document Identification, Authorization, Work Management Data.

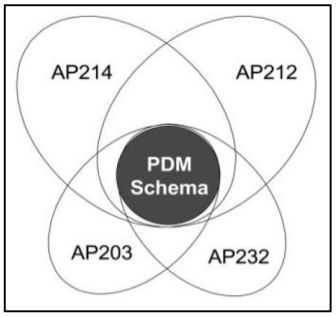


Figure 61: PDM schema scope and positioning

ISO 10303-242 (STEP **AP242**) or “Managed model based 3D engineering" is the merging of the two leading STEP application protocols for Mechanical Design: Aerospace's STEP AP203 and Automotive's STEP AP214. Both standards are widely used in the supply chains of many industrial sectors and they share common data structures. Rather than pursue costly parallel development and maintenance efforts, developing a new convergent STEP standard has been proposed. The major technical impact of the STEP AP 242 standard covers the following areas: Model-Based Development, PDM integration and PDM services, Long Term Archiving, supply chain integration, engineering design data exchange includ-

ing composites, advanced Product Manufacturing Information, mechatronics and requirement management. STEP AP 242 Edition 1 provides all the functionalities covered by the AP 203 ed2 and AP 214 ed3.

As shown on opposite Figure 62, the scope of AP242 includes also new functionalities, such as the “Shape Quality” modules and its “Product data quality business object model”, a new model for STEP 3D tessellated geometry (e.g. laser scanning, …), capabilities for STEP “external element references” used for kinematics and other disciplines such as CAD 3D construction history / parametric / 3D assembly constraints).

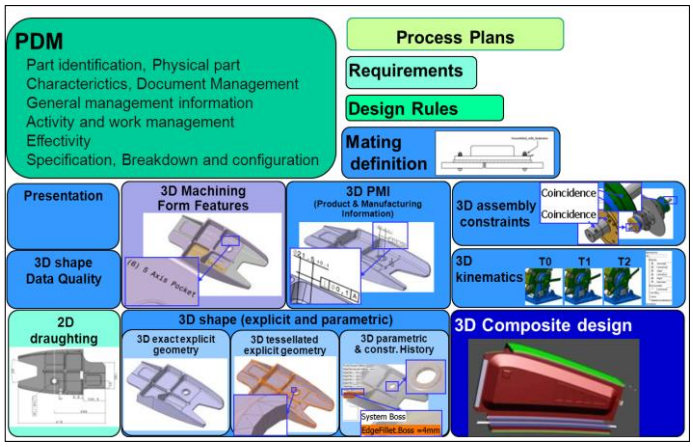


Figure 62: Overview of AP242 scope and content

Figure 63 below provides an overview of the current AP242 business object model capabilities.

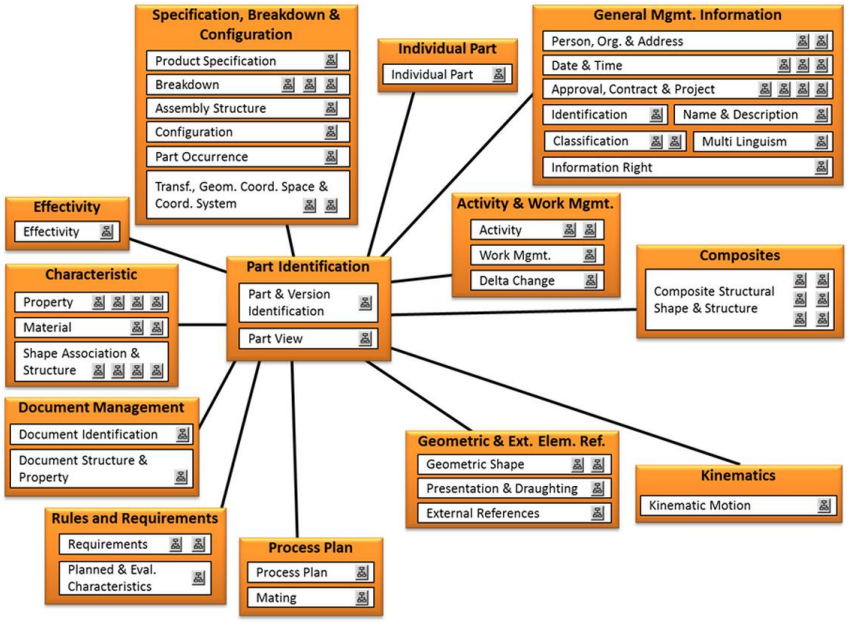


Figure 63: Overview of AP242business object model capabilities

8.1.1.3 STEP-based product and part identification and definition

The **Product** entity represents the product master base information in STEP. According to ISO/TS 10303-1017:2010, a Product is the identification of a product or of a type of product. This entity collects all information that is common among the different versions and views of the product. In the STEP PDM Schema, a general product can be conceptually interpreted either as a part or as a document. In this way, parts and documents are managed in a consistent and parallel fashion [ProSTEP_iViP, 2002]. As illustrated by the taxonomy in Figure 64 below, the PLCS defines other subtypes of the product entity: it can be whether a part, a system, an interface, a breakdown, a requirement, a document or a slot.

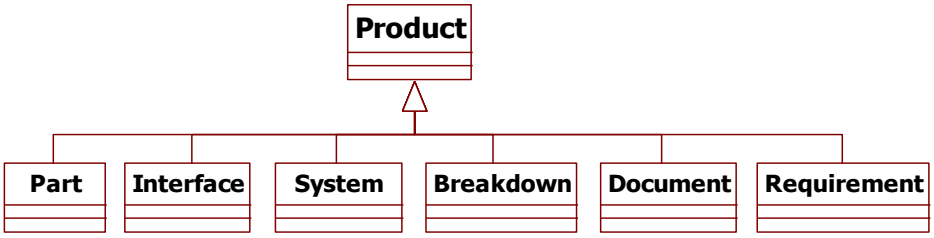


Figure 64: Product sub-types entities in the PLCS

According to ISO/TS 10303-1017:2010, a **Part** is a discrete object that may come into existence as a consequence of a manufacturing process. The Part entity is a sub-type of product that contain all information related to the successive versions of a part or a product constituent that cannot be dismantled. The UoF "Part Identification" of the PDM schema is the centre for assignment of further product management data. It provides the information model to capture the various product/part definitions and the link to the related properties and representations as shown on Figure 65 below.

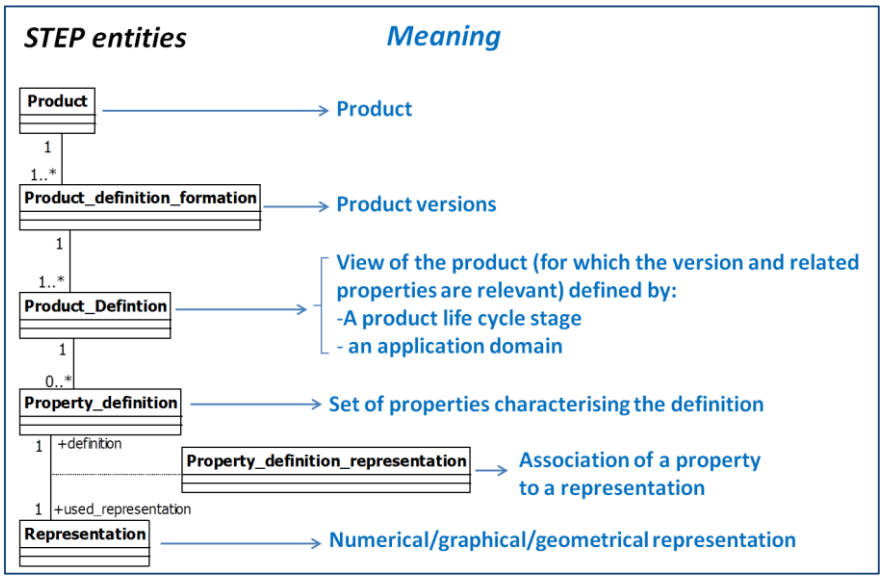


Figure 65: Part identification schema and the meaning of STEP entities

AP 203 has a rule (product_requires_version) which requires that all part products be associated with a **Product_definition_formation** entity supporting the versioning of parts. A Product has one or more **Product_definition_formation** or **Product_definition_formation_with_specified_source** which corresponding technical definitions used in specific application context are captured by the Product_definition entity. One definition is related and characterised a set of properties (Property_definitions) which are associated to their corresponding **Representation** with the entity **Property_definition_representation**. This is the semantic used in Part 41, in AP203 and in the PDM schema.

AP214 and AP239 use other semantics but the data structure remains approximately the same. For instance, in the PLCS, a **product_definition_formation** becomes a **product_version** and the **product_definition** becomes a **product_view_definition** defined by a **view_definition_context**. In the AP214 a part is identified by the entity **Item** instead of Product. An Item is either a single object or a unit in a group of objects. It collects the information that is common to all versions of the object. An Item may be either a single piece part, an assembly of arbitrary complexity, a raw material, or a tool. An item version can have multiple applicative views: **Design_Discipline_Item_Definition**, each having one or more **Application_Context** (defined by a life cycle stage and an application domain). A **Design_discipline_item_definition** is a view of an Item version relevant for the requirements of one or

more life cycle stages and application domains. This view collects product data for a specific task. For each usage of a part in a product assembly an **Item_Instance** is created. These instances refer to a definition related to a view of the item version. An Item instance is the occurrence of an object that is defined either as a **Design_discipline_item_definition** or as a **Product_Specification**. Each Item instance is a Single instance, a Quantified instance, a Selected or Specified instance. Each of these instances may carry additional information like placement or its relevance for a specific product configuration (effectivity). Interchangeable parts (design alternatives or alternate parts) can be identified with the entity **Alternate_Item_Relationship**, and the dependencies between various item versions can be captured with the entity **Item_Version_Relationship** (see Figure 66).

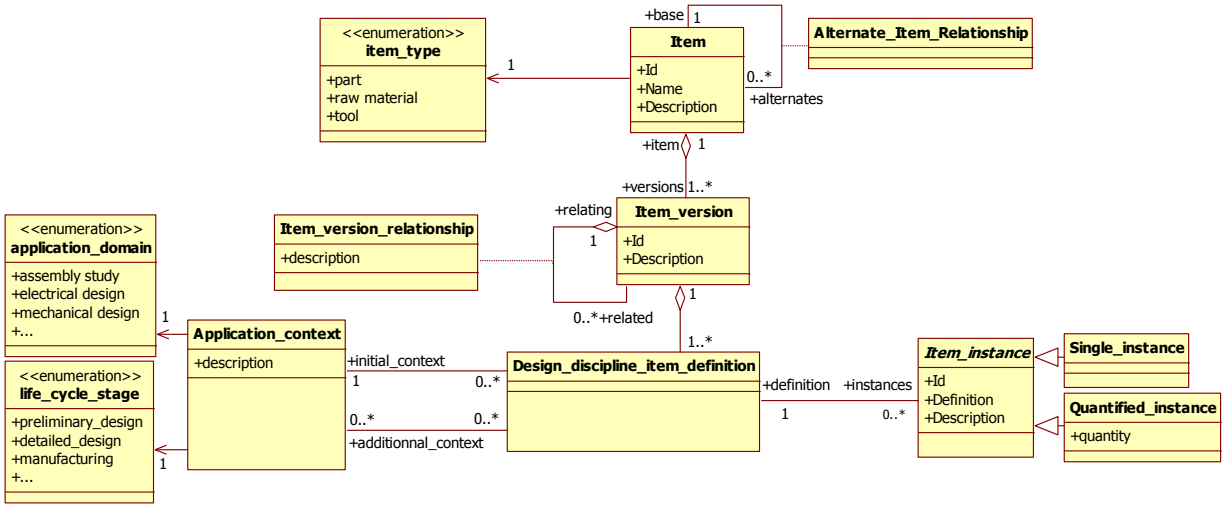


Figure 66: UML representation of the AP214 "Part Identification" UoF

8.1.1.4 STEP-based part properties and shape representations

The PDM Schema based on part 41, allows specifying properties associated product data and parts definitions by linking a representation of the property values to the object with which the property is associated. A **property** is the definition of a special quality and may reflect physics or arbitrary, user defined measurements. A number of pre-defined property type names are also proposed for use when appropriate (recyclability property, mass property, quality property, cost property and duration property). A **representation**, in the context of part properties, is a collection of one or more **representation_items** related to a **property_definition** through the entity **property_definition_representation** (see Figure 67 below).

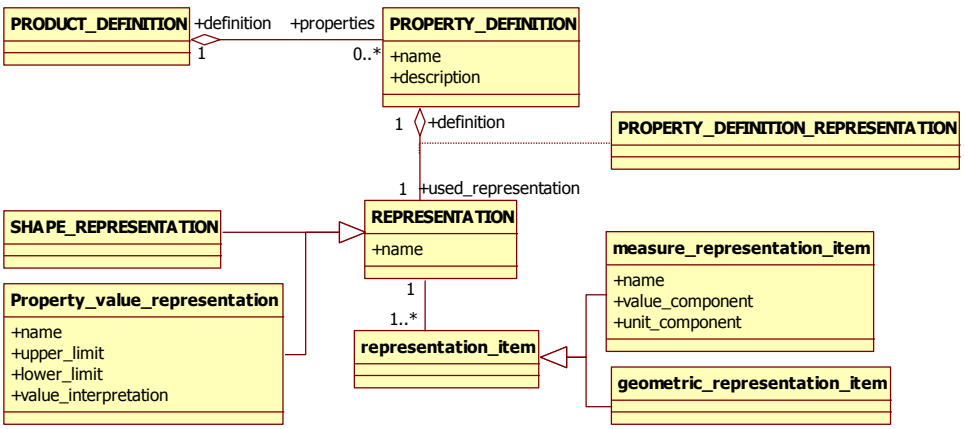


Figure 67: Property Definition Associated with Product Data in the PDM schema

The sub-types of representation_items used to describe the property depend in function of the type of representation used (i.e. shape_representation or property_value_representation). For instance, a **measure_representation_item** is a value element that participates as an item in one or more property_value_representation. The **representation_context** defines the context of interpretation for the values of items in a representation. The PDM schema uses the same item_property types than AP214 but does not include material properties. In AP214, an EXPRESS "Select data type" entity called "**Item_information_select**" is associated to the **Design_discipline_item_definition** entity and enables to select the relevant properties that characterise the definition of the item version (see Figure 68 below). The UML language/representation used in Figure 68 does not allow managing "Select data type" entities. Specific inheritances and operations must be defined to get the equivalent principle in UML. Three sub-types of Property are defined in the AP214:

- **Item_property**: An Item property is a characteristic of an item. Each Item property is either a General item property, a Recyclability, an Item cost property, a Mass, a Material property or an Item quality property;
- **Shape_dependant_property**: it is a characteristic of an object that is closely related to the shape of the product/item. In the ideal case most of the Shape_dependent_property objects can be derived from the geometric shape representation. Each Shape_dependent_property refers to zero or one **Item_shape** (corresponding to the Shape_Representation entity in AP203 or in the PDM schema);
- **Process_property**: it is a characteristic of a process related object.

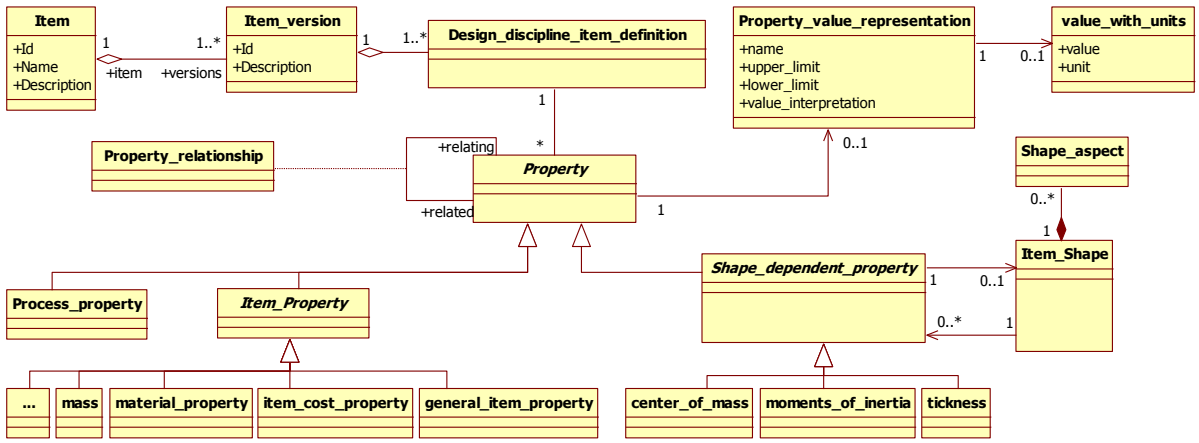


Figure 68: UML representation of the AP214 "Item_property" UoF

In STEP, a special case of part properties is that of the part shape property - the representation of the geometrical shape model of the part or item. Part geometry is identified in the PDM Schema as a representation of a property of a part definition. The external part shape is represented by the entity **shape_representation**, a subtype of **representation**. As with general part properties, a representation of a property is identified and linked to the product definition by the entity property_definition. As shown Figure 69, for the part shape property, property_definition is specialised to the subtype **product_definition_shape** and **property_definition_representation** is specialised to the sub-type **shape_definition_representation.** The **product_definition_shape** may be a conceptual shape for which a specific geometric representation is not required [ProSTEP_iViP, 2002].

STEP Application Protocols define various subtypes of **shape_representations** with differing constraints on the allowable **representation_items** to explicitly represent the detailed geometric model. This geometry may be defined in STEP format as well as in native CAD format. Certain detailed elements of the shape are required to be able to place and relate the external geometric models together.

Therefore, placement information is placed in the set of items of each **shape_representation**. Placement information is modelled using the entity **axis2_placement_3d**, a subtype of **geometric_representation_item** that specifies the location and orientation in three-dimensional space of two mutually perpendicular axes that can be used to define transformations between **shape_representations**.

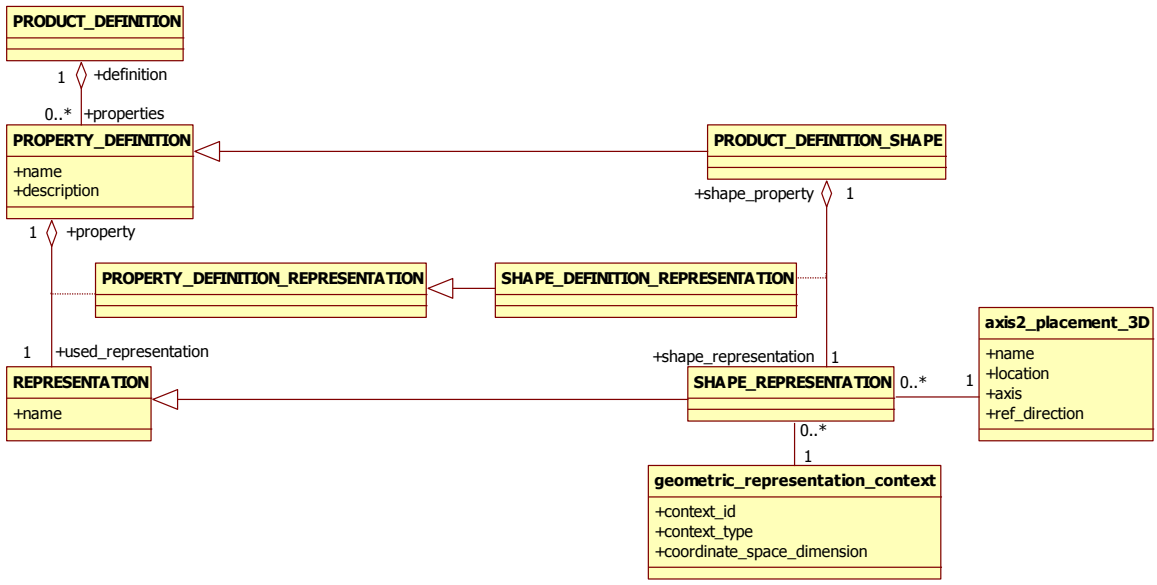


Figure 69: UML schema of part geometric properties as managed within the PDM schema

The shape information represented is either the complete shape of a part or corresponds to a specifically identified portion of a part shape model are as shape_aspects. Typically the geometric elements that establish the **shape_aspect** are collected in a **shape_representation** which is related to the **shape_aspect** via instances of **property_definition** and **shape_definition_representation** as shown on Figure 70 below.

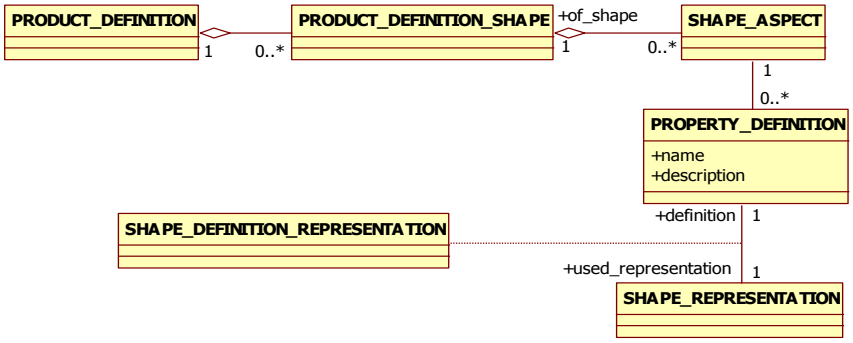


Figure 70: shape and shape aspects association

In STEP, shape representations are managed as part meta-data that does not typically represent the detailed geometry of the part shape. The PDM Schema rather recommends to link instances of **shape_representation** to the corresponding part master data. The external part shape is an external geometric model related to the part master data and is referenced as an external CAD file [ProSTEP_iViP, 2002]. External files are managed in STEP through the Document_Identification UoF according to the fundamental STEP "Document as Product" principle. The "Document as Product" approach consists in identifying and managing documents PDM objects like the products or parts objects. In Figure 71, the Document entity inherits from the Product entity which means that a **Document** has one or more **Document_versions** (product_formation) that are characterized by a set of **properties**

defined by several **representations.** A representation, in the context of document properties, is a collection of one or more **descriptive_representation_items** (content, creation properties, format, etc.) or **measure_representation_items** (file size, page count, etc.). A document representation definition may optionally be associated with one or more constituent external files (**document_file**) that make it up. External files in the PDM Schema represent a simple external reference to a named file. The external file that contains the detailed geometry is attached to the part master data in two ways (see Figure 71):

- Related to the **shape_representation** representing the external geometric model using the entities **property_definition** and **property_definition_representation** (Figure 71a);
- Related to the product identification data using **applied_document_reference** (Figures 71b1 and 71b2).

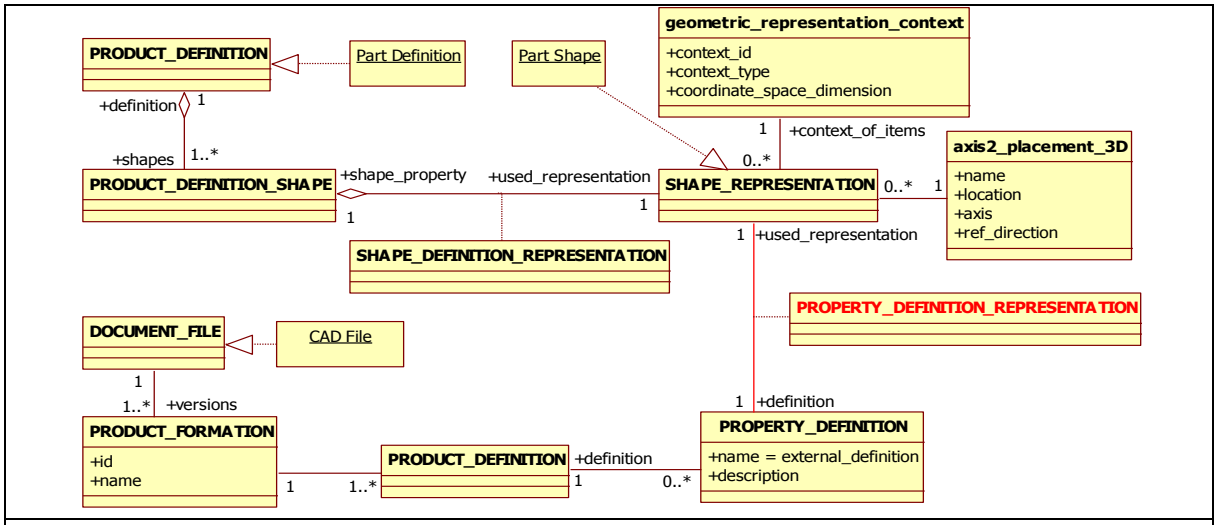


Figure 71a - Geometric model file related to part identification via property_definition_representation

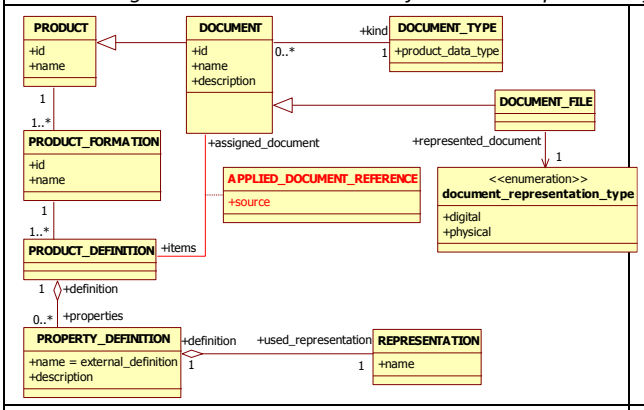


Figure 71b1 - Geometric model file related to part identification via applied_document_reference

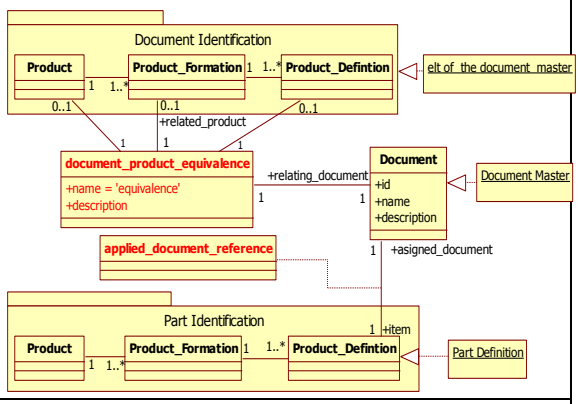


Figure 71b2 - Relating a part definition to a constituent file of a document master

Figure 71: the PDM schema methods to relate a geometric model file to product identification data

The PDM schema generally recommends using the entity **applied_document_reference to relate an external data file to** the **product_definition** of a part identification. However several scenarios exist:

- If the **document_file** representing the CAD model exists alone as an unmanaged external file reference, the **applied_document_reference** relates the **product_definition** of a part identification directly to the **document_file** representing the CAD model.
- If the **document_file** representing the CAD model exists as a constituent file of a managed document (using the "Document as Product" approach), the document_reference should be applied from the managed document via the **document_product_equivalence** entity which

asserts that the related product (or product_definition_formation or product_definition) represents an element of a document master that is assigned as a reference to some other product data.

Various relationships are defined between external geometrical models to represent geometric model structure and the associated transformation information required for digital mock-up of assembly structures. These aspects are explained in the next section below.

8.1.1.5 STEP-based configured DMU product structures

This section describes how STEP proposes to manage hierarchical product structures representing assemblies and the constituents of those assemblies.

A **product structure**, as defined in [Fischer&Sachers, 2002], can contain abstract components, assemblies and parts (also software) of a product or component. It can be expressed through various representations. In the AP214 CC8 recommended practices [Fischer&Sachers, 2002], authors distinguish two types of product structure to manage Digital Mock-Ups:

- **Abstract product structure**: based on abstract/logical/generic/conceptual product components or functions, which each serves as a "placeholder" for one or more representations of physical components (technical solutions). Abstract product structures usually contain configuration information (specifications and configuration rules);
- **Explicit assembly/product structure**: that contains all non-variant sub-assemblies and parts of a product.

A **component** is defined in [Fischer&Sachers, 2002] as an object within a product structure that is part of a product. Components can represent abstract **product_component** or **product_function**, or physical components like non-variant explicit assemblies or parts (items). AP214 CC8 defines a product components taxonomy including the following notions and entities:

- **Product Class**: A Product_class is the identification of a set of similar products to be offered to the market. The top level element of a product_class structure is a product_component identified as *root_entry_for* the product_class.
- **Product_Complex**: It is an object with the capability that it can be realized by, decomposed into or specialized as Product_constituent objects in a functional, logical, or physical way. Each Complex_product is a Product_function, a Product_component, or an Alternative_solution.
- **Product_Constituent**: A Product_constituent is an object that may participate in the functional, logical, or physical breakdown or be an alternate realization of a Complex_product. Each Product_constituent is a Product_component, a Product_function, or an Item_instance.
- **Product_Component**: A Product component is an element in a product decomposition structure. A Product component is represented by a set of alternate Item solution (see Figure 72) objects with common functional requirements. The top level Product component of the decomposition tree shall be associated to a Product class as root entry. The corresponding decomposition structure is identical for all variations of all products of that Product class.
- **Product_Function**: A Product function is the mo de of action or activity by which a product fulfils a certain purpose.
- **Item_Instance**: An Item_instance is a sub-type of product_constituent acting as the occurrence of an object that is defined by a Design_discipline_item_definition.
- An **Alternative_solution** or **Item_solution** is the identification of one of potentially many mutually exclusive implementations of a Product_function or of a Product_component.

However the multi-inheritances of product_component and product_function which are both complex_product sub-types and product_constituent sub-types are difficult to implement. That is why, most implemented conformance classes do not use these entities and recommend to use and instantiate their sub-types (product_component, product_function, technical solution and item_instance). AP214 hence supports two kinds of product breakdown:

- Structural breakdown: a hierarchical structure whose nodes are the Product_Component.
- Functional breakdown: a hierarchical structure whose nodes are the Product_Function.

As shown in Figure 72, different kind of relationships can be established between these breakdowns and their elements through the use of **Product_structure_relationship** and its sub-types (decomposition, functionality, realisation, derivation). A Product_structure_relationship relates a Product_Complex (function, component or solution) to a Product_constituent (function, component or item_instance). [Chambolle, 1999] defines the various types of relationships that can be established between these entities and as well as the related implementation methods for managing these relationships:

- Functionality: relationship permitting to allocate a product_function to a product_component or to a product_class, corresponding to the entity function_component_association;
- Decomposition: relationship permitting to define hierarchical links between product_functions (for functional breakdowns) or between product_components (for structural breakdowns);
- Realisation: relationship enabling to associate a set of potential alternative_solutions to the corresponding product_components or item_instances that the solution implements; it corresponds for instance to the entity solution_instance_association;
- Derivation: relationship permitting to relate various alternatives solutions and get the traceability of the successive design alternatives that have been proposed for a given solution.

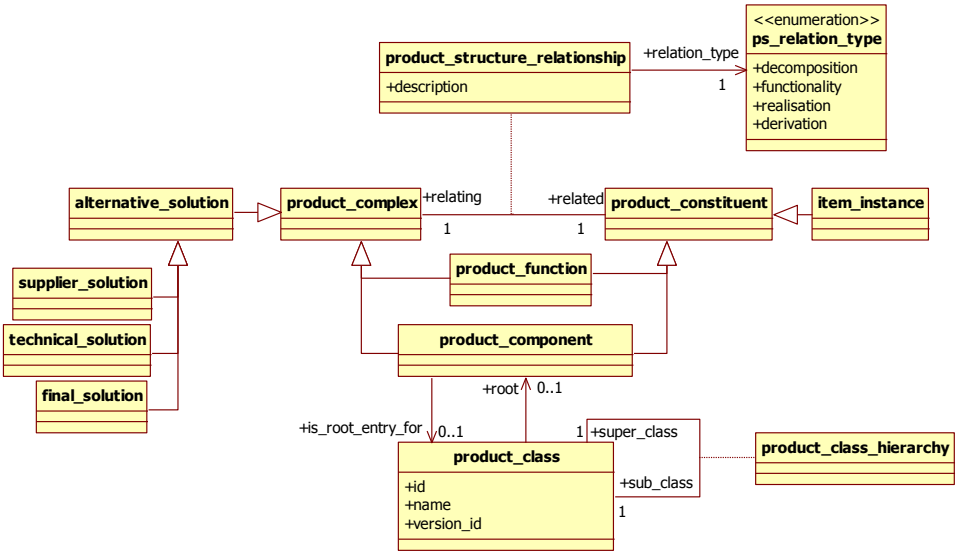


Figure 72: Product structure elements relationships as defined in [[ISO, 2000b], Fischer&Sachers, 2002]

Concerning structural assemblies - whose structures involve hierarchical relationships between **product_components** - two similar methods can be found in STEP to define the product breakdown nodes relating a parent assembly to its related constituents:

- Parent-children relationships are made between **product_definition** entities representing a view definition of the part master through the use of the sub-types of the **assembly_component_usage** entity. In that case, the relationship itself represents the usage occurrence of a constituent definition within the immediate parent assembly definition (see Figure 73). This method is the most commonly used in the majority of STEP APs.
- Parent-children relationships are made between the **Design_discipline_item_definition** of the parent assembly and the **Item_instances** used as parts or components of this assembly through the use of a sub-type of the **item_definition_instance_relationship** entity: **assembly_component_relationship**. In that case, the item_instance represents the usage occurrence of a Design_discipline_item_definition within the immediate parent assembly definition (see Figure 74). This method has been used only in STEP-AP214.

These two methods are similar since, in both cases, the **assembly_component_usage** or the **assembly_component_relationship** permit to relate the **product_definition** or **design_discipline_item_definition** of the parent assembly to the **product_definition** or **design_discipline_item_definition** of its constituents. The difference is that in the first method, the product_definition usage occurrence is represented by the **assembly_component_usage** relationship, whereas in the second method it is represented directly by the **Item_instance** entity. This difference comes from the fact that the second method proposes to manage both abstract and explicit product structures where the item_instances are used as realisation of one of the variants of an abstract product_component.

STEP has also the capability to identify individual occurrences of component in a multi-level assembly. This provides the ability to assign to each occurrence an identifier, a position in the assembly, a geometrical representation, or other properties that may be different from that assigned to the part definition of the component.

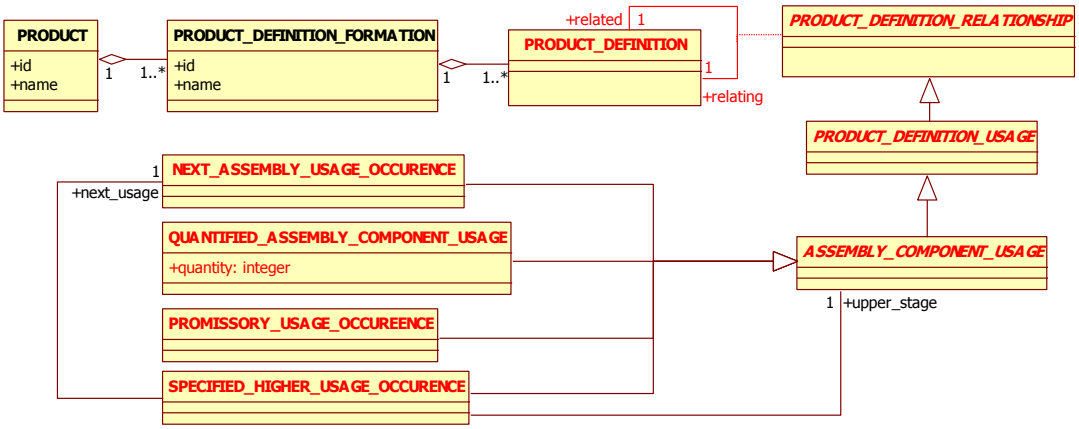


Figure 73: Assembly_component_usage relationship representing the usage occurence of a product_defintion within the immediate parent assembly definition accordig to the PDM schema

The PDM schema does not recommend to instantiate the assembly_component_usage entity as itself, but to instantiate its subtype **next_assembly_usage_occurrence (NAUO)**. This subtype represents a unique individual occurrence of a component definition as used within the parent assembly. As used in an immediate next higher parent assembly. The id attribute contains a unique instance identifier for the individual component occurrence and can correspond to the identifier of a functional item relationship. Other assembly_component_usage sub-types are defined:

- **quantified_assembly_component_usage**: It adds the attribute quantity to identify the number of occurrences of the constituent definition that are used in the parent assembly. This

entity should be instantiated when quantities greater than one need to be represented, but when the individual occurrences do not need to be independently distinguished. This is created as a "complex" instance of next_assembly_usage_occurrence AND quantified_assembly_component_usage.

- **promissory_usage_occurrence**: It represents the usage occurrence of a component within a higher-level assembly that is not the immediate parent, in the case where the detailed assembly structure in between the component and the higher-level assembly is not represented. A promissory_usage_occurrence specifies the intention to use the constituent in an assembly. It may also be used when the product structure is not completely defined.
- **specified_higher_usage_occurrence**: It represents the specific use occurrence within a higher-level assembly of an individual occurrence of a component definition used in an immediate parent sub-assembly.

The other method (as defined in AP214 CC8) uses **product_components** and **product_structure_relationships** to describe the common decomposition structure of all products of a product class; i.e. the abstract product structure. **Item_solutions** are used to describe the variants for a **product component** and **item instances** are used to identify elements of an **item_solution** that are used in an explicit assembly structure. Figure 74 illustrates the distinction and the links between entities permitting to manage explicit assembly structures (red area) and entities permitting to manage abstract product structures (green area).

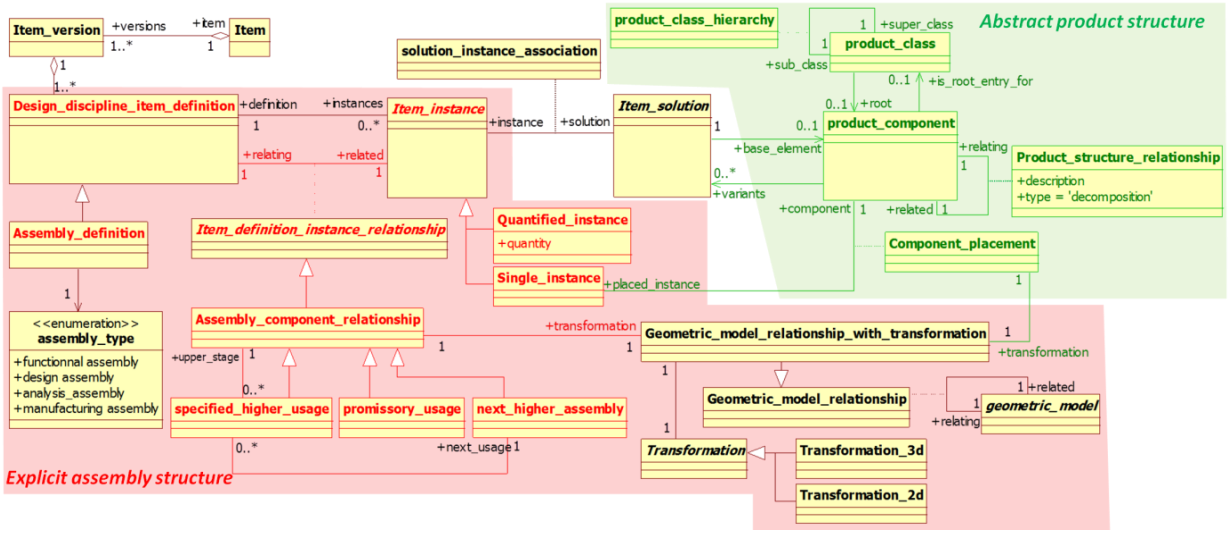


Figure 74: Abstract Vs Explicit product structure information model as defined in AP214 CC8

In AP214, a **Variant** is defined as an alternative solution for an abstract product component or product function. A variant is a technical solution that fulfils the functional requirements of a function or abstract product component in a certain way. The abstract product component includes all variants. In AP214 a **Configuration** is a part/product structure that represents a valid product variant. The instances of the parts in a specific configuration can be physically composed to a valid manufactured product. For representing the configuration information, in AP214 the configuration object is used to associate product classifying objects with rules and product structure in order to determine which objects are valid in the context of a certain product class (configuration information). Therefore, and as shown on Figure 76, configuration information is the information required to identify the relevant variants (**item_solutions** and related **item_instances**) of abstract **product_components** that are used in

an explicit assembly structure of a **product_class**. A configuration may point to all levels of **product_component** or **item_ instances**.

The PDM schema, based on the integrated resource Part 44 [ISO, 2004a]– Product structure configuration – defines the **configuration_item** entity which represents the identification of a particular configuration, i.e., variation of a **product_concept** (equivalent to **product_class** in AP214). A **configuration_item** is defined with respect to the product concept, i.e., the class of similar products of which it is a member. The **configuration_item** defines a manufacturable end item, or something that is conceived and expected as such. The valid use of component parts for planned units of manufacturing of a particular **configuration_item** may be specified using the concept of "effectivity". **Effectivity** is a generic concept defining the valid use of the product data to which the effectivity is assigned. It is hence the designation that something or a relationship between two things is used or planned to be used in some **configuration_item**. Within the PDM Schema, there are two areas identified for the usage of the "effectivity concept" [ProSTEP_iViP, 2002]:

- **Configuration_effectivity**: in that case effectivity is designated on relationships between product_definitions and are restricted to be assigned to a particular usage occurrence of a constituent part in some higher-level assembly (see Figure 75).
- **General_validity _period_effectivity**: restricting the domain of applicability of the associated product data to a date range, a particular lot or serial number range, or to a time interval respectively. In that case effectivities can be assigned more generically to a much wider variety of product data by using the **applied_effectivity_assignment entity**.

Figure 75 provides a UML class diagram of the configuration management schema as defined in Part44 [ISO, 2004a]. Figure 75 only illustrates the usage of **configuration_effectivity** applied on the sub-types of the **product_definition_usage** entity.

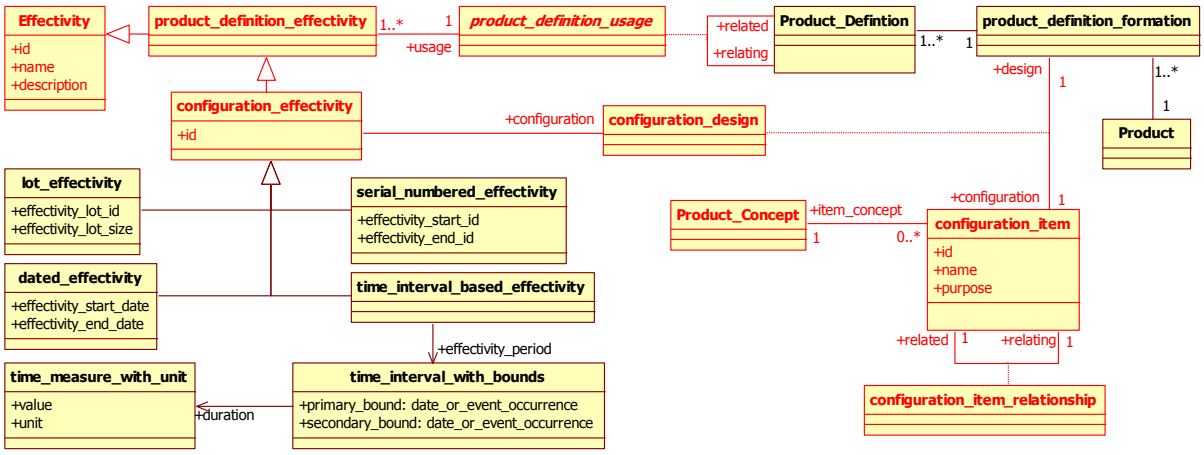


Figure 75: UML representation of the configuration management schema as defined in [ISO, 2004a]

As shown on the above class diagram, **effectivity** is designated on relationships between **product_definitions** by either range of serial numbers, ranges of dates or a lot. This is accomplished through a complex instance of **configuration_effectivity** and one of either **serial_numbered_effectivity**, **dated_effectivity**, **lot_effectivity** or **time_interval_based_effectivity**. It should be noted that, in the ISO standard, these entities are immediate sub-types of the **effectivity** entity. However, translating the express-defined standard in UML, this model corresponds to the correct data structure. A **serial_numbered_effectivity** specifies an effectivity_start_id with an optional effectivity_end_id. If the effectivity_end_id does not exist, the effectivity is good for the starting serial number and all following serial numbers. A **dated_effectivity** follows the same pattern using dates rather than serial numbers. A

lot_effectivity indicates an effectivity_lot_id and an effectivity_lot_size. These entities are related to a **product_definition_relationship** through the **usage attribute** in the **product_definition_effectivity** entity. The **configuration_effectivity** entity relates these relationships to a **configuration_design** which generally relates a **configuration_item** to a **product_definition_formation**. This does mean that all configuration_items must be associated to a design version in order to have effectivity. It should be noted that, in STEP, all effectivities are explicit and there are no assumed effectivities. If a part is effective for all instances of a product model, the data should explicitly state all the effective instances [ISO, 2004a].

Figure 76 shows how explicit assembly structures may be related by documents enclosing geometry models for the related parts, assemblies and/or sub-assemblies. These documents can be connected with each other representing the document structure for the appropriated assembly structure. Each **item_instance** used in an assembly, as well as each corresponding **assembly_component_relationship** that permit to relate it to the parent assembly definition, must carry placement information defined by transformation matrixes positioning the **item_instance** in the reference frame of the parent assembly. It is also possible to integrate placements of abstract product components by transformation matrices the entity **Component_placement** or **Instance_placement** (depending the edition of AP214) can be used. It specifies the relationship between two **Product_components** added by placement information. A **Component_placement** is the information pertaining to the placement of a **Product_component**, which is defined in its own **Cartesian_coordinate_space** or in the coordinate space of a reference **Product_component**.

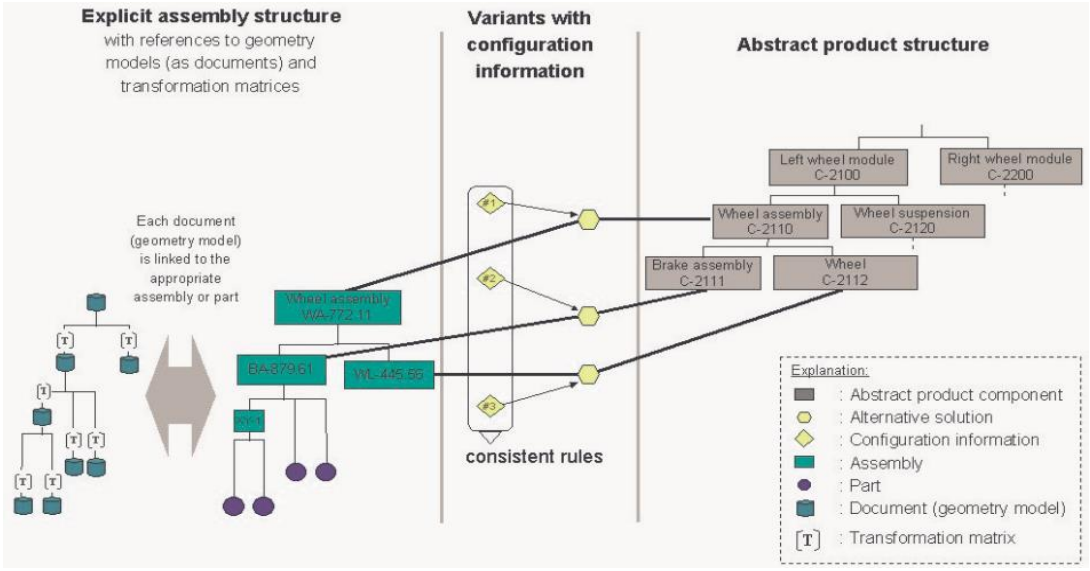


Figure 76: From Abstract product structure to Explicit configured and positioned DMU structure [Fischer&Sachers, 2002]

The PDM Schema, based on part 43 (“Representation Structures”), allows linking geometric structures that result from relating different **shape_representations** with associated product structure when applicable. The UML class diagram of the Representation_Schema of Part 43 is given in Figure 77.

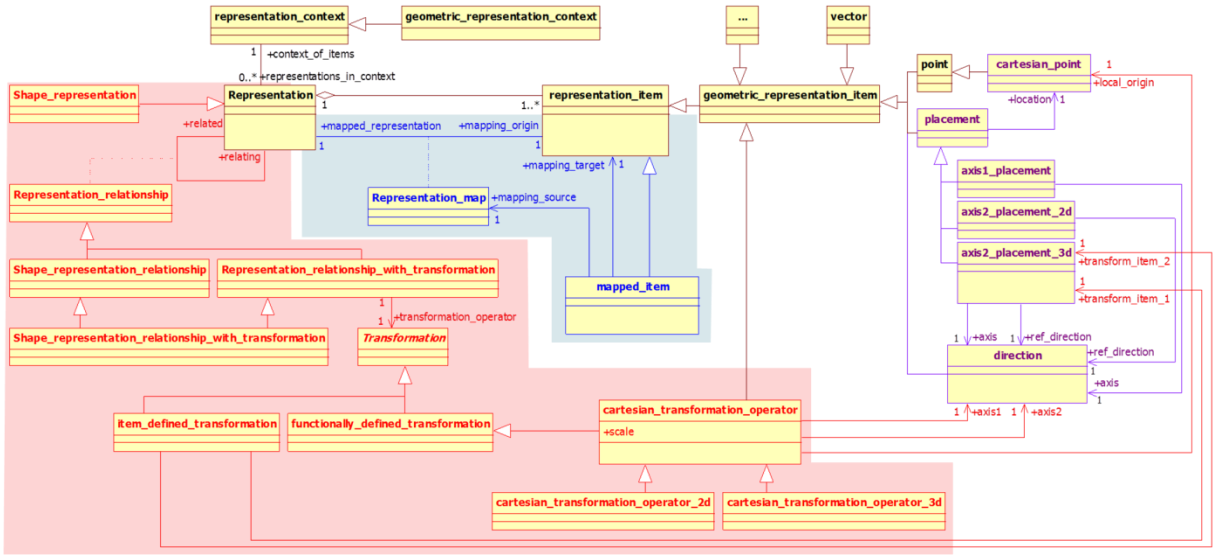
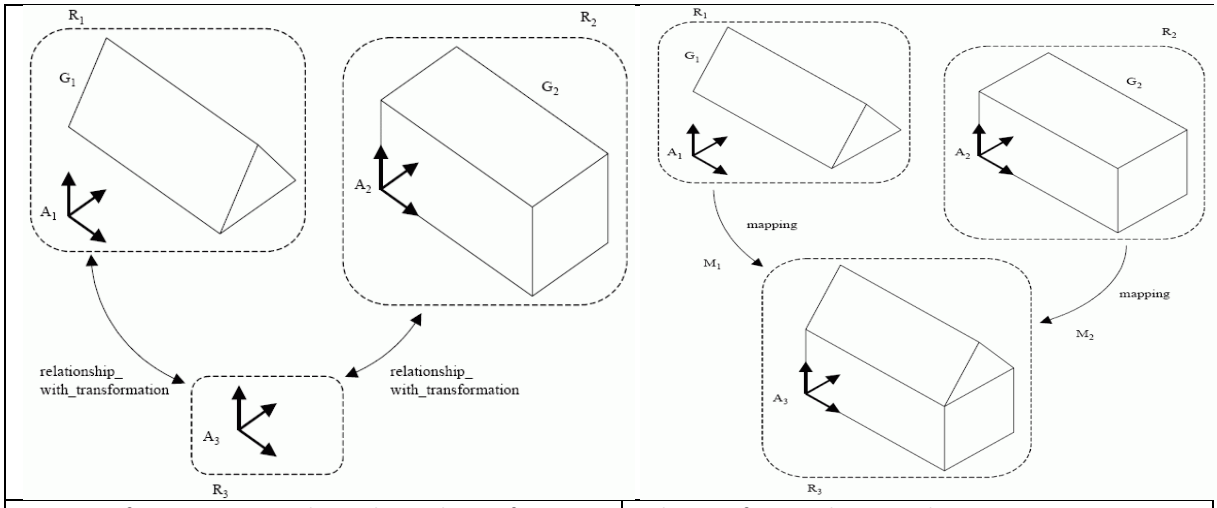


Figure 77: UML class diagram of the STEP Representation Schema as defined in Part 43 [ISO, 2011a]

There are two methods that may be used to relate a component part's shape to the shape of the assembly in which it is assembled:

- The first method (red area in Figure 77) consists of defining the shape for each part (component and assembly), and then relating the two shapes and providing the information that defines the orientation of the component part with respect to the assembly part through a transformation. This method shall be used to relate the shapes that are represented by different representation types.
- The second method (blue area in Figure 77) consists of defining the shape for each part (component and assembly), and then incorporating the shape of the component directly in the shape of the assembly. This method may be used if the types of the components representation within the assembly's representation are the same.

Figure 78 illustrates the use of both methods on a house/building assembly example extracted from ISO 10303-43 [ISO, 2011a].



78a- Use of representation relationship with transformation | 78b- Use of mapped_item and representation_map

Figure 78: Comparison of the two methods for relating a component part's shape to the shape of its parent assembly

When using the first method (Figure 78a), each of the **shape_representation** entities that define the shapes of the component and assembly **product_definitions** is related through references in the

shape_representation_relationship entity. In this case, orientation information is provided by a "complex instance" of **shape_representation_relationship** and **representation_relationship_with_transformation** entities. The **representation_relationship_with_transformation** entity references a **transformation** allowing the orientation to be defined using an **axis2_placement_3d** entity in each representation for an **item_defined_transformation** or a **cartesian_transformation_operator** entity for a **functionally_defined_transformation**. On Figure 78a, three instances of **representation** are shown. The first representation R1 (the roof) contains the geometry G1 and an **axis2_placement_3d** A1 and the second representation R2 contains the geometry G2 and an **axis2_placement_3d** A2. Two instances of **representation_relationship_with_transformation** allow R1 and R2 to be associated with a third **representation**, R3 representing the shape of the building assembly. In that case, R3 contains a single item: an **axis2_placement_3d**. The associations between R1 and R3, and between R2 and R3, do not make R1 and R2 parts of R3. However, the associations between R1 and R3, and between R2 and R3, allow an application to infer that G1 and G2 can be combined and used to describe the shape of the building assembly.

When using the second method (Figure 78b), the **shape_representation** entity that is referenced by an instance of a **representation_map** entity that is referenced by the **mapping_source** attribute of an instance of the **mapped_item** entity. The attribute **mapped_representation** of the **representation_map** will reference the **shape_representation** subtype that defines the geometric and/or topological representation of the shape. The instance of the **mapped_item** entity is then added to the set of items in the **shape_representation** entity that defines the geometry of the assembly. Figure 78b shows how the **mapped_item** and **representation_map** entity data types can be used to describe the composition of one representation from other instances of representation. In that case, two instances of **representation_map** allow R1 and R2 to be used as elements in a third representation, R3 representing the shape of the assembly. Respectively, the instances of **representation_map** RM1 and RM2 reference R1 and R2 as their **mapped_representations** and A1 and A2 as their **mapping_origins**. R3 contains as its items an **axis2_placement_3d** and two instances of **mapped_item**, M1 and M2. Respectively M1 and M2 reference RM1 and RM2 as their mapping_source and they share a common mapping_target: A3. The result is that R3 uses R1 and R2 as parts of its definition.

The usage of both alternatives is considered reasonable, because both mechanisms make sense even in mixed combinations. With regard to the transformations in the context of assembly, a part is in principle incorporated in the assembly only by rigid motion (i.e., translation and/or rotation) [ProSTEP_iViP, 2002]. In AP214, a **Transformation** is a geometric transformation composed of translations and rotations which are defined based on the reference frames (placement entities) of the shape_representations of the component and of its parent assembly. Each Transformation is either a **Transformation_3d** or a **Transformation_2d** depending on the coordinate space dimension.

8.1.1.6 The Core Product Model and its extensions

In addition to these standards, the NIST [Sudarsan et al., 2005b] propose a product information modelling framework to support the full range of PDM/PLM information. This framework intends to capture product data, design rationale, assembly, tolerance information, product evolution and, product families. It is based on the NIST **Core Product Model** (CPM) and its extensions. The CPM is a generic, abstract product meta-data model with generic semantics, defined as a UML class diagram (shown in Figure 79) that gives equal status to three aspects of a product or artefact: its function, form and behaviour. Therefore, the key object in the CPM is the "**Artifact**" that represents a distinct entity in a product, whether that entity is a component, part, subassembly or assembly.

CPM consists of two sets of classes, called object and relationship classes. There are five abstract classes (classes that cannot be instantiated but their sub-types) from which all CPM objects or relationships inherit [Fenves et al., 2007]:

- **CoreProductModel** represents the highest level of generalization;
- **CommonCoreRelationship** is the base class for all association classes;
- **CommonCoreObject** is the base class for all object classes;
- **CoreEntity** is the base class from which **Artifact** and **Feature** are specialized;
- **CoreProperty** is the base class from which **Function**, **Flow**, **Form**, **Geometry** and **Material** are specialized.

There are four relationship classes [Fenves et al., 2007]:

- **Constraint** is a specific shared property of a set of entities that must hold in all cases. In CPM, only the entity instances that constitute the constrained set are identified;
- **EntityAssociation** is a set membership relationship among artifacts, features and ports;
- **Usage** is a mapping from **CommonCoreObject** to **CommonCoreObject**, particularly useful when constraints apply to the specific "target" entity but not to the generic "source" entity, or when the source entity resides in an external catalog or design repository;
- **Trace** is structurally identical to Usage, particularly useful when the "target" entity in the current product description depends in some way on a "source" entity in another product description. The type attribute of Trace specifies the nature of the dependence (alternative_of, version_of, derived_from, is_based_on, etc.).

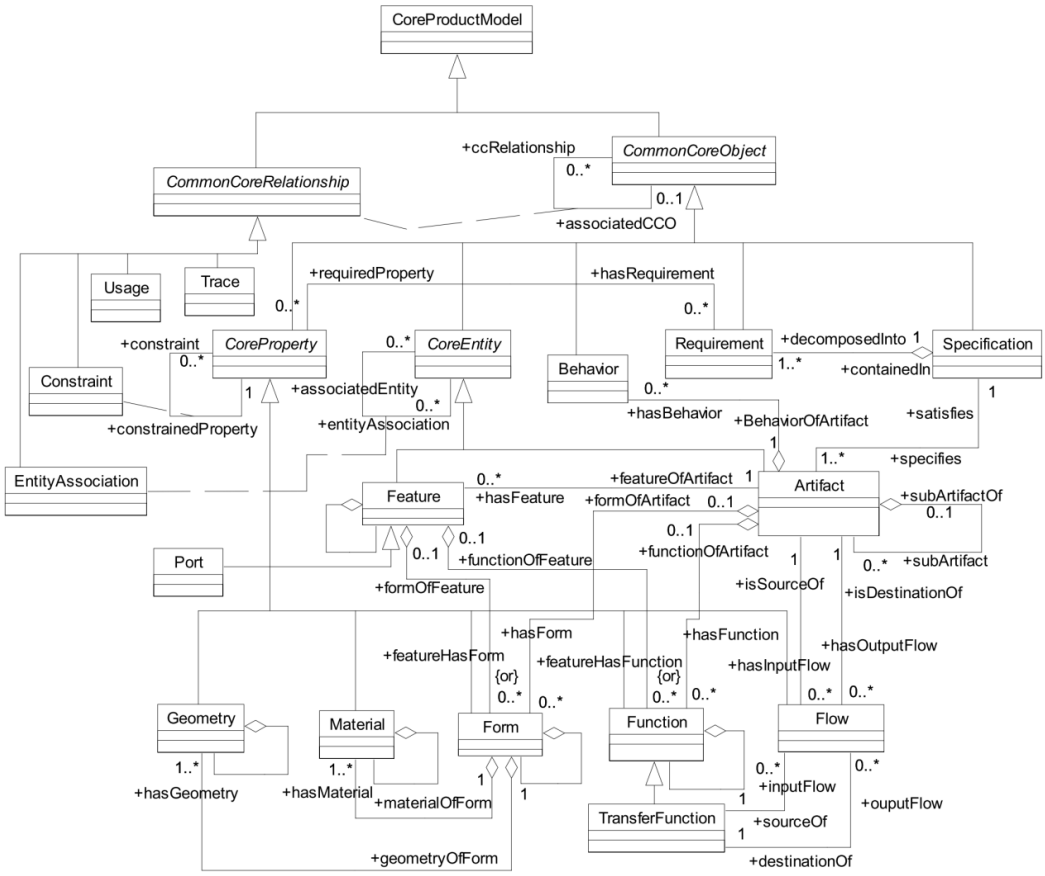


Figure 79: UML class diagram of the Core Product Model from [Fenves et al., 2007]

We will not detail all CPM objects and relationships definition but certain aspects of this data model will be referenced in the following sections because some data structures of this model have

proven their relevance in the area of product data management and for addressing issues that are common to our research work. The initial objectives of the CPM was to provide a common data model among four in-house research and development projects at NIST as well as a base-level data representation for a multilevel design information flow model [Fenves et al., 2007]. The first version of the Core Product Model (CPM) responding to these objectives was presented in [Fenves, 2002]. The objective therefore became to expand the CPM to serve as the basic, top-level model for all product realization information [Fenves et al., 2007] leading to some CPM extensions among which:

- the **Open Assembly Model** (OAM) provides a standard representation and exchange protocol for assemblies [Sudarsan et al., 2005a, Sudarsan et al., 2006]. The assembly model defines both a system level conceptual model and the associated hierarchical relationships. The model provides a way for tolerance representation and propagation, kinematics representation, and engineering analysis at the system level.
- the **Design-Analysis Integration project** proposes a conceptual data architecture that can provide tighter integration of spatial and functional design and support analysis-driven design and opportunistic analysis [Fenves et al., 2003]. CPM serves as the organizing principle of the Master Model from which discipline-specific functional models (views) are idealized.
- the **Product Family Evolution Model** extends CPM to the representation of the evolution of product families and of the rationale of the changes involved [Wang et al., 2003]. The model represents the independent evolution of products and components through families, series and versions, and the rationale for the changes with a case study on the CFM aero-engines product family evolutions.

Extensions and implementations of CPM may explicitly assign attributes to specializations of the CPM objects and relationships so as to provide interoperability with new systems, legacy data models such as STEP, or existing CAD programs. The OAM and the Design-Analysis Integration conceptual data models are respectively addressed in sections 8.1.2 and 8.1.3.

8.1.2 Capturing product assembly and interfaces information in PDM systems

Since the design of complex and large engineering systems is increasingly becoming a collaborative task among distributed design teams, exchanging parts and assembly information between modelling and analyses systems is critical for unrestricted exchange of product data. However, little has been done in terms of developing standard representations that specify assembly information and knowledge [Sudarsan et al., 2006]. An assembly information model, according to Sudarsan *et al*, contains information regarding parts and their assembly relationships. As mentioned in section 7.2.2, the preparation of a large assembly simulation model compared to a standalone component implies a preparation process of interfaces connecting components together. Therefore, assembly relationships concern not only hierarchical links between parts or sub-assemblies instances within various product structures, but also the interfaces objects specifying how components are connected together. Previous section has permitted to understand how the STEP standard proposes to model hierarchical relationships. This section provides an overview of current **existing data models enabling to capture interfaces and interaction objects specifying how components are functionally and physically connected together**.

8.1.2.1 The MOSAIC project and related Sellgren's works

Within the MOSAIC project [Andersson&Sellgren, 1998], authors developed an object-oriented approach to FEA modelling where products are considered as **systems** described by recursive sub-systems and related through **interfaces**. The modelling paradigm defines a system model as an idealized representation of a system at a level of complexity and detail to complete analysis. As represented in EXPRESS-G in Figure 80, system models are aggregated models that can be decomposed into models of sub-systems, interfaces that connect the sub-systems, and an **orientation** object that defines the spatial orientation of the system. Each interface is an aggregation of two **mating faces** that are related to two different sub-models [Sellgren&Drogou, 1998]. A sub-model can have a finite set of mating faces.

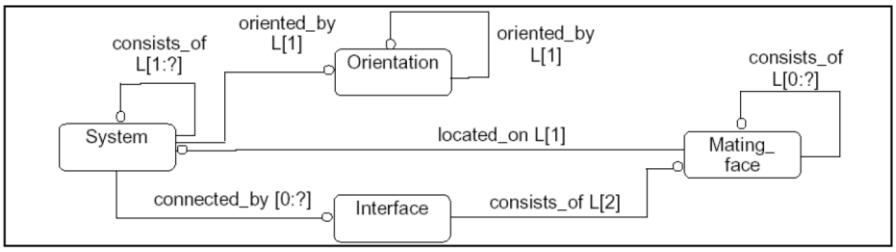


Figure 80: A Systems Model and its Relationships [Sellgren&Drogou, 1998]

As shown in Figure 81, the building blocks of sub-models are referred to **Behaviour features** that represent a **form features** at a specific level of abstraction for a specific physical domain. According to authors, a **behaviour feature** should describe the physical properties and behaviour of an object independent of how it will be connected to other features. Authors also state that "due to the strong relationship between shape and behaviour, it is strongly desirable that objects in the two domains have and associativity relation, i.e. that the CAD and behaviour domains are integrated". A **mating face** is an object that reference discrete FE entities in a behaviour feature. A **mating face** is a geometric **face** located at the boundary of a **form feature** (defining a **shape)** and may exist at different levels of abstraction.

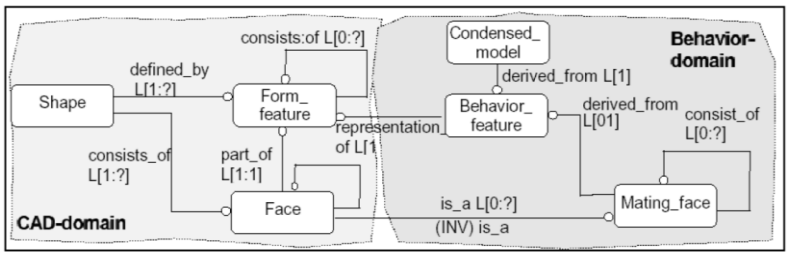


Figure 81: Behaviour Feature and Mating Face Associated to Design Shape [Sellgren&Drogou, 1998]

The FEA mating relations are treated as relations between nodal degrees of freedom (DOFs) in two different bodies. The sub-models may be compatible or incompatible. Incompatible sub-models make it very difficult to mesh transitions. The interface between models can be specified as contact or attachment. To deal with incompatible bodies, the nodal relations are based on the master and slave node concept (see Figure 82 below).

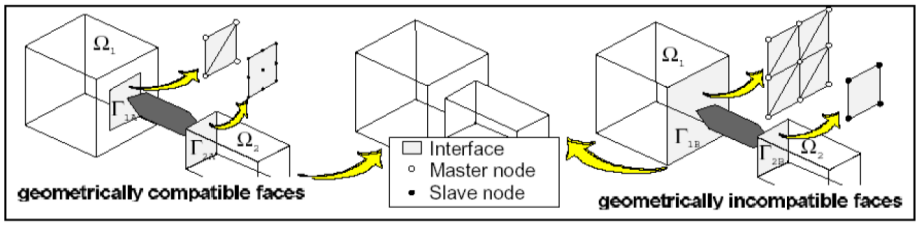


Figure 82: Implicit Master-Slave Selection [Sellgren&Drogou, 1998]

A few years later, in [Sellgren, 2006a], the author highlights the need to rely on a modular model architecture that enables configuration of systems models from a stored library of sub-models and interface models. He also proposed a model-based and feature-based interface information model as extension and improvement of PDM/PLM data models [Sellgren, 2006b, Sellgren, 2009]. This information model aims at linking design and behavior assembly models by proposing a three-layered architecture data model (see Figure 83) that include interface objects.

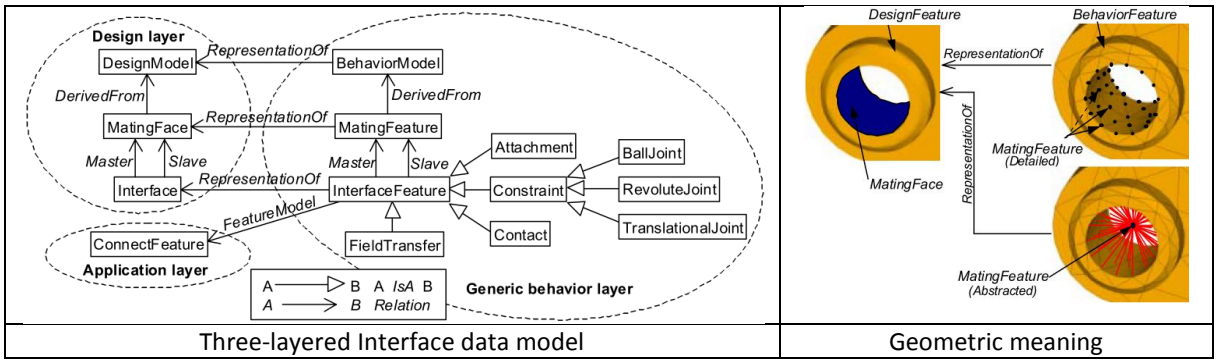


Figure 83: Interface modelling as a 3-layered architecture: design, behaviour, and applicative layers [Sellgren, 2006a]

The design layer manages design models (i.e. CAD models) and related features such as shape, material, and orientation in space. The interactions between design sub-models take place at interfaces, where an interface is a pair of mating faces. The “generic behavior layer” manages behaviour models designed to represent a specific behaviour of a design model. Within these two layers, mating features are related to design models’ mating faces as their discrete representations in the behavioural model. An interface feature is the discrete representation of an interface at a generic behaviour level. The application layer permits to relate an interface feature (i.e. actual connection of two behavioural sub-models) to its equivalent meaning and representation in FE software-specific proprietary formats. As shown in Figure 84, Sellgren integrates these concepts within a more complete product information model so that these metadata can be managed within PDM/PLM applications. In order to interoperate with CAX and PDM applications, sub-models’ features (such as material, node numbers, property ranges) and connect features are stored in source files that are self-contained models in proprietary format.

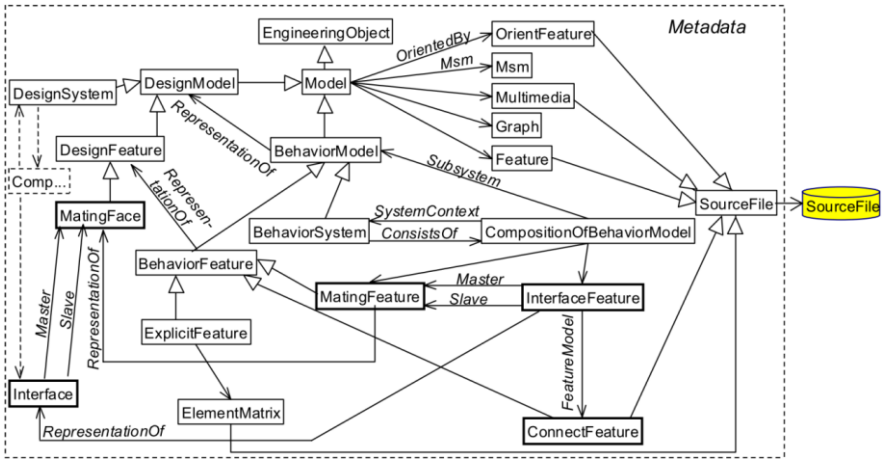


Figure 84: Interface model objects put into the context of a larger product information model [Sellgren, 2006a]

8.1.2.2 STEP-based interface data models

In AP214, the **mated_item_association** entity is a sub-type of **item_definition_instance_relationship**. But instead of representing a hierarchical link between two item_instances, it represents a physical mating link between two product constituents, that is to say two item_instances. Figure 85 provides the corresponding UML class diagram showing the difference between an assembly definition defining the hierarchical link between two item_instances and a mating definition (in red) defining the physical mating link between two item_instances.

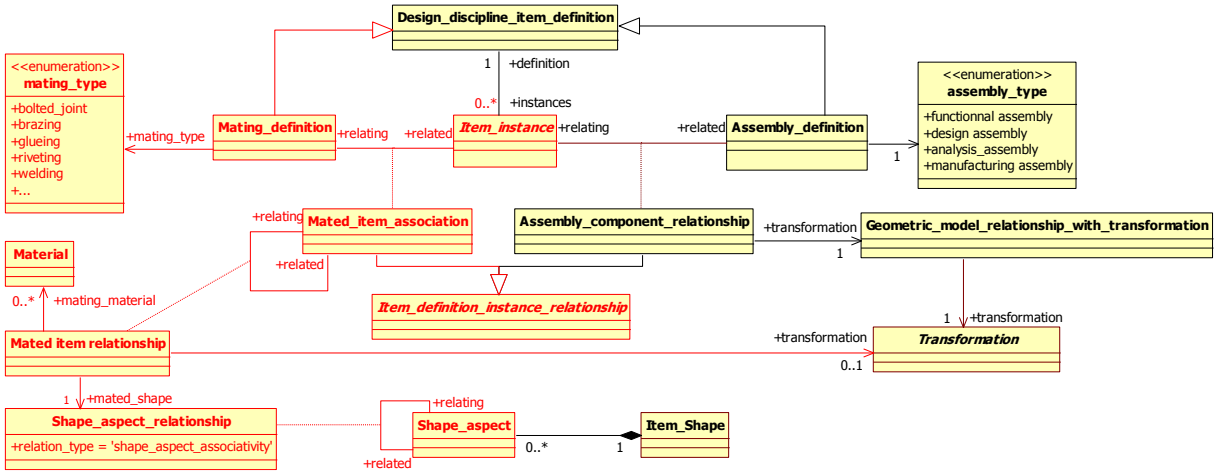


Figure 85: Mating definition as defined in ISO 10303-214

A **Mating_definition**, as sub-type of **Design_discipline_item_definition**, is a view of an **Item_version** defining the physical connection of two or more **Item_instance** objects including technical information about the kind of connection. This information is independent from the hierarchical assembly structure. A **Mated_item_association** allows specifying the involvement of an **Item_instance** within a **Mating_definition**. A **Mated_item_relationship** is a relationship between two **Mated_item_association** objects specifying additional information about the mating of two particular **Item_instances** that go into a Mating definition. The two **Mated_item_association** objects that are referenced by the **Mated_item_relationship** refer to the same **Mating_definition**. The **mated_shape** specifies the **Shape_aspect_relationship** that relates the two **Shape_aspect** objects that form the area of mating contact.

The integrated application resources **ISO 10303-105** (“Kinematics”) and **ISO 10303-109** (“Kinematic and Geometric Constraints for Assembly Models”) permit to capture respectively the assembly

constraints (specifying explicit geometric constraints between item instances) and the kinematic links and constraints between two adjacent components at a joint. The kinematic structure schema in ISO 10303-105 defines the kinematic structure of a mechanical product in terms of links, pairs, and joints. The kinematic motion schema in ISO 10303-105 defines kinematic motion used to represent the relative motion between assembly components. The related used entities are **kinematic_pair** representing the geometric aspects of the kinematic constraints of motion between two assembled components and the **KinematicPath** representing the relative motion between assembly components.

The assembly_feature_relationship_schema defined in ISO 10303-109 provides resource constructs for bridging **shape_aspect_relationship** with its **representation_relationship** and for detailing geometric information of the **representation_relationship**. Detailed geometric relation between two components can be represented via necessary number of pairs of assembly features one belonging to one constituent and the other belonging to the other constituent. This enables feature level correspondence between constituents. In most assembly related applications, not only correspondence of assembly features but also more detailed geometric constraint information (such as parallelism, coincidence, tangency, co-axial) are required in geometric entity level. These geometric constraint specifications applied between two constituents are summarised in the assembly_constraint_schema [ISO, 2003b].

The terms used in AP233 related to interfaces and connections does not align exactly with the terms used in other standards. In AP233, an Interface connector is the term for the part of a system that interacts with other systems or the environment and the Interface connection is the link between connectors. The connectors correspond to the concept of “Ports” as defined in object-oriented modelling approaches. This is illustrated in the Figure 86.

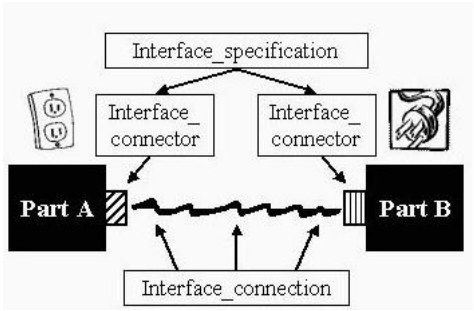


Figure 86: AP233 interface connection and connectors concepts

These concepts are supported by the information model shown on the UML class diagram of Figure 87.

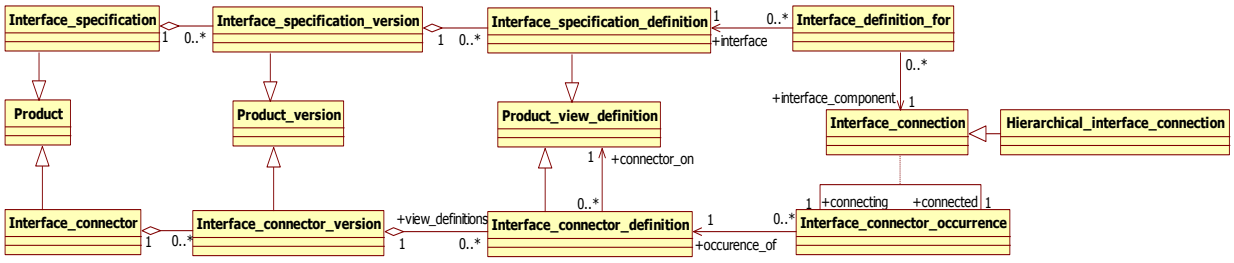


Figure 87: Interface information model as defined by [ISO, 2012]

An **Interface_connector** is a specialization of **Product** that identifies a part of a product with which one or more other products or the environment interacts. An **Interface_connector_definition** is a specialization of **Product_view_definition** that identifies a view of an **Interface_connector_version**. The *connector_on* attribute references the item/component for which the **Interface_connector_definition**

provides an interface capability. An **Interface_connector_occurrences** is an occurrence of an **InterfaceConnectorDefinition**. Each **Interface_connector_occurrence** represents the place where a product used in an assembly can interact with other products in the assembly. The interaction is represented by an **Interface_connection** that relates a connected pair of **Interface_connector_occurrences**. An **Interface_specification** is a specialization of **Product** that provides a definition of necessary attributes for one or more items that participate in an interface. The **Interface_specification** specifies an **Interface_connection** that conforms to the specification and the relationship is captured through the use of the **Interface_definition_for** entity. A **Hierarchical_interface_connection** is a specialization of **InterfaceConnection** that provides an interconnection between components at different levels in an assembly. Each connection point in the assembly is represented by an **InterfaceConnectorOccurrence**.

8.1.2.3 The Open Assembly Model (NIST)

In [Sudarsan et al., 2005a, Sudarsan et al., 2006], a group of researchers working for the NIST have proposed a standard representation and exchange protocol for assembly and system-level tolerance information: the Open Assembly Model (OAM). This model incorporates tolerance representation, kinematics, assembly relationships, and assembly features. In the OAM, an **Assembly** is a composition of its subassemblies and parts. A **Part** is the lowest level component. The OAM is based on the data structure of the NIST Core Product Model since the Assembly and Part entities inherit function, behaviour, and form from the Core Product Model's **Artifact** class. The OAM assembly data structure for specifying explicit geometric and kinematic constraints between artefacts uses the data structures of STEP (mainly ISO10303-105, ISO10303-108 and ISO10303-109). Figure 88 shows the main UML class diagram of the OAM. This diagram schema incorporates information about assembly relationships and component composition through the use of the class **AssemblyAssociation**. An **AssemblyAssociation** represents the component assembly relationship of an assembly. It is the aggregation of one or more **ArtifactAssociation**.

An **ArtifactAssociation** class represents the assembly relationship between one or more artifacts. **ArtifactAssociation** is specialized into the following classes: **PositionOrientation**, **RelativeMotion**, and **Connection**. **PositionOrientation** represents the relative position and orientation between two or more artifacts that are not physically connected and describes the associated constraints between the artifacts. **RelativeMotion** represents the relative motions between two or more artifacts that are not physically connected and describes the associated constraints between the artifacts.

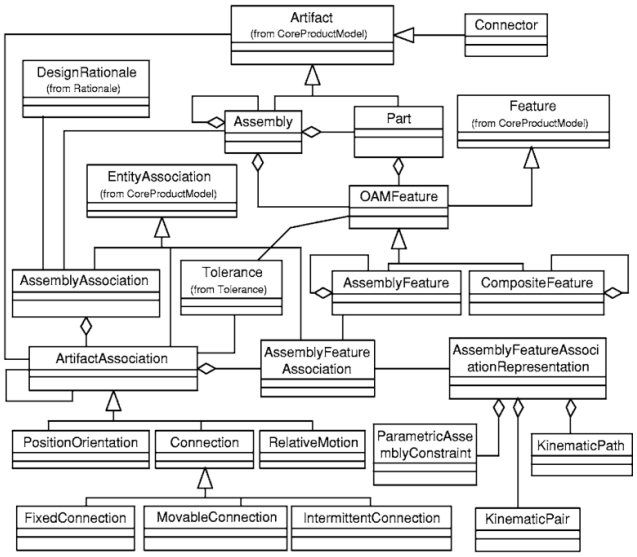


Figure 88: Class diagram of the open assembly model from [Sudarsan et al., 2006]

Connection represents the connection between artifacts that are physically connected. Connection is further specialized as **FixedConnection**, **MovableConnection**, or **IntermittentConnection**. **FixedConnection** represents a connection in which the participating artifacts are physically connected and describes the type and/or properties of the fixed joints. **MovableConnection** represents the connection in which the participating artifacts are physically connected and movable with respect to one another and describes the type and/or properties of kinematic joints. **IntermittentConnection** represents the connection where the participating artifacts physically connect only intermittently (e.g., cam). **Connector** realizes **Connection**, which is a specialization of the **Artifact**.

Each assembly component, whether it is a sub-assembly or a part, is made up of one or more features represented in the model by **OAMFeature**. **OAMFeature** is a subclass of the CPM Feature class. **AssemblyFeature**, a subclass of **OAMFeature**, represents assembly features. Assembly features are a collection of geometric entities of artifacts. They may be partial shape elements of any artefact. The class **AssemblyFeatureAssociation** represents the association between mating assembly features through which relevant artifacts are associated. The class **ArtifactAssociation** is the aggregation of **AssemblyFeatureAssociation**. An artefact association is the aggregation of assembly feature associations. Any assembly feature association relates in general to two or more assembly features (except in the case where an artefact association involves only one artefact; it may involve only one assembly feature). The class **AssemblyFeatureAssociationRepresentation** represents the assembly relationship between two or more assembly features. This class is an aggregation of ParametricAssemblyConstraints, a KinematicPair and/or a KinematicPath as defined in ISO10303-105.

8.1.2.4 The MUVOA model

The “MUlti-View Assembly Oriented” (MUVOA) data model presented in [Demoly, 2010, Demoly et al., 2011c] support the definition, specification and capture of various product-component relationships. Figure 89 shows an extract of the MUOVA UML class diagram. This model encompasses four kinds of product-component relationships:

- Contact relation: physical contact relation between two components;
- Precedence relation: assembly logical order for two components in contact and in non-contact;
- Kinematic relation: additional information on contact relation which enables the description of constrained degrees of freedom (rotation and translation) for each part of the product;
- Technological relation: additional information on contact relation which enables the definition of the “assemblability” of the product, and therefore on the mating relation between two components in contact.

These relationships are aggregated into “Bill of Relations” to describe an “assembly configurations” that will serve to automate the generation of assembly sequences and skeleton for providing the contextual design of preliminary design CAD models. Assembly constraints, assembly interface features and related geometric specifications are as well specified and captured within this model. Appendix VIII (and related Figure 250) provides the complete MUVOA model UML class diagram.

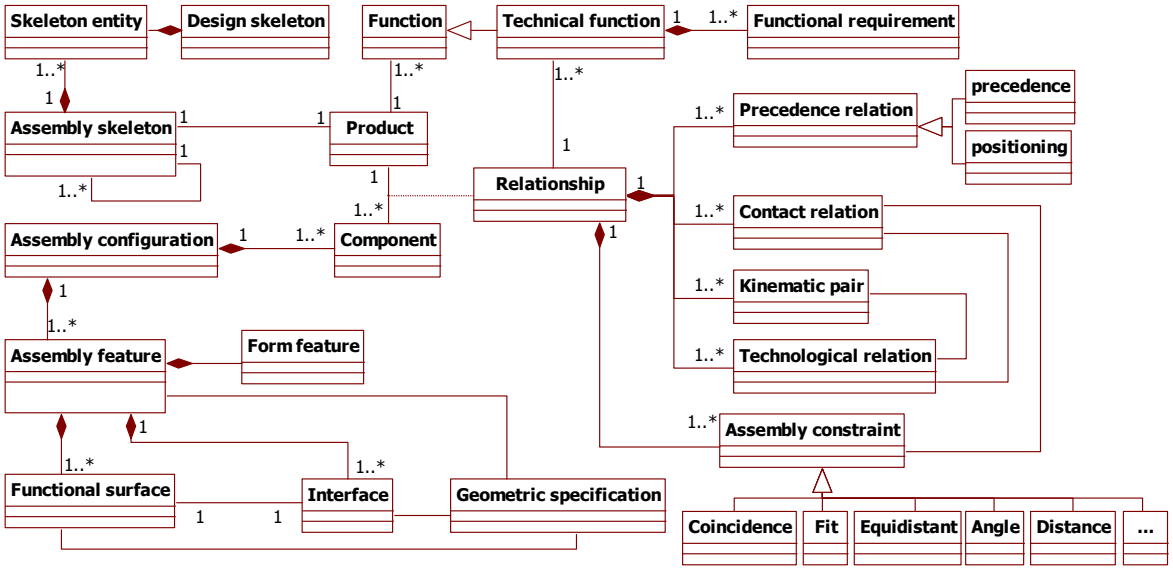


Figure 89: Product-component relationships within the MUOVA data model redesigned from [Demoly et al., 2010b]

8.1.3 Continuity of design and simulation data in product data models

This section focuses on the formal description of frameworks for integrating CAD and FEA information and on product data models enabling to support the related integration capabilities. Such capabilities concern the sharing of product information within design-simulation loops, traceability of related activities inputs and outputs and the enhancement of re-use of appropriate design and analysis artefacts regarding simulation objectives and related modelling requirements.

8.1.3.1 Frameworks for integrating CAD and FEA activities and data

In the early 1990's, Arabshahi, et al. presented a vision of CAD and FEA based design-analysis integration [Arabshahi et al., 1991]. The authors develop an IDEF0 process model of the FEA process. As a natural extension to this work, [Arabshahi et al., 1993] present the activities of an automated CAD-FEA transformation. The aim of the work is to enable the analysis of the product to respond to design changes and allow seamless integration between design and analysis. The proposed framework aims at automating the creation of analyses models from the design model. The framework consists of the following:

- A Product Description System (PDS) to hold the geometric data and non-geometric data associated with the product;
- A semi-automatic means for transforming the geometric and non-geometric data to an analysis model that can be meshed;
- Intelligent meshing routines to provide varying degrees of meshing and feedback on the meshed geometry;
- A series of finite element solvers for a range of solutions;
- A post-processing capability to associate results from the idealized model to the design model and allow for modifications.

In summary, Arabshahi, et al. developed a vision for integrating CAD and CAE. Although technology has increased greatly, the problems that existed 20 years ago are still present in the current state of engineering analysis.

In [Belaziz et al., 2000], authors develop a feature-based tool to aid in the integration of analysis during design. The tool is based on the morphological analysis of solid models, and a simplification and idealization process. Modifications can be made to the design features based on parameterization. It is possible to walk back from the idealized model to a new modified solid model based on analysis results. The morphological analysis concept is based on the idea that an object is created from a solid "stock" through a progression of modification steps. The morphological features are classified into elementary features, composite, interacting, and characteristic relationships. The morphological analysis is completed in three steps: (1) detect all characteristic modifications; (2) re-constitute the previous model based on the modifications; and (3) code the modifiers. The structure of the morphological analysis is included in Figure 90.

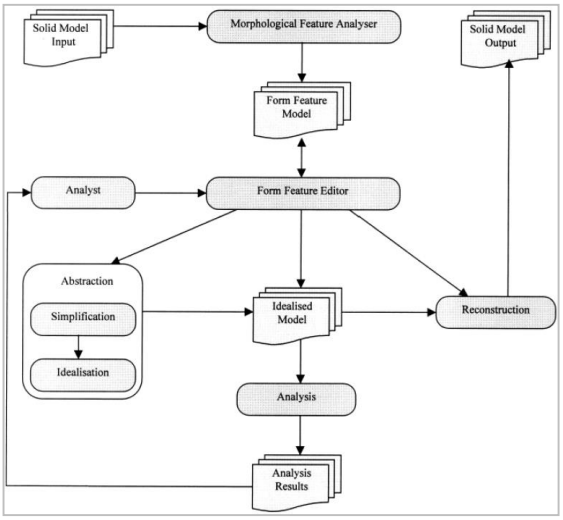


Figure 90: Morphological Analysis Tool Components [Belaziz et al., 2000]

The form feature model can be obtained in two ways. If the geometry exists, the features can be mapped to the geometric model. This allows each feature to be associated with a particular function. If the geometry does not exist, designers can create a feature description. Next, the analysis model is generated in a two-phase process of simplification and idealization. In the simplification phase, irrelevant information is cleaned out of the model. In the idealization phase, the geometry is constructed to ideal shapes. The analysis is then completed on the idealized model. Finally, reconstruction enables the recreation of a solid model based on analysis results. This idea supports the bi-directionality needed for design-analysis integration. The reconstruction allows modification made during the analysis phase to propagate to the design phase.

More recent research works on design-analysis integration focused on modelling knowledge capitalisation in order to enhance re-use of design-analysis artefacts as well as the appropriate modelling methodologies. [Troussier, 1999], [Peak et al., 1999] and [Bellenger et al., 2008] formalized simulation objectives and hypotheses applied to a design model when setting up simulations to capitalize and reuse them in future model preparations.

In [Mocko et al., 2004], authors define a knowledge representation and proof-of-concept repository implementation for capturing and sharing engineering behavioural models. The goal is to develop a clean graphical user interface that permit to reduce the knowledge gap between engineering design and analysis by facilitating the reuse of behavioural models. To achieve this, authors propose a meta-data representation for formally characterizing behavioural models. The meta-data representation captures the assumptions, limitations, accuracy and context of engineering behavioural models. Based on this knowledge representation, a proof-of-concept repository is implemented for archiving and exchanging reusable behavioural models. This “knowledge repository” allows designers and analysts to select behavioural models that are appropriate for their desired simulation context and understand the underlying assumptions and limitation of the model. The reuse of behaviour models can be increased while reducing the risk of misuse because validated behavioural models and the associated application context are published to the repository.

In [Badin, 2011, Badin et al., 2011], authors proposed a method of knowledge management used in several interacting activities within a design process. There, analysts and designers collaborate and exchange design information. However, the authors assume that relationships between dimensional

parameters of CAD and CAE/FEA models of components are available, which does not currently exist. Additionally, they refer to configurations where the shapes of components are identical in the design and simulation contexts.

8.1.3.2 CAD-CAE integration in product data models

The Design/Structural Analysis integration problem is typified by the requirement to share geometric shape and analysis information in an iterative environment. Most CAE systems currently employ point-to-point translators to facilitate the connection between design and analysis. However, point-to-point translation does not solve all CAD-CAE integration issues, because it does not contain the full richness of the product information originally associated with the solid model. A standards-based data model is required to enable this integration and hence reduce the number of translators [Hunten, 1997].

The **ISO 10303-209** ("Composite and Metallic Structural Analysis and Related Design") has been developed to address this approach to the Design/Structural Analysis problem. The approach used in AP209 is to define and integrate separate product definitions for the analysis and design disciplines [Hunten, 1997]:

- **The analysis discipline product definitions** concern finite element models, analysis controls, and analysis outputs. Loads and boundary conditions may be applied to either mesh or geometry. Linear statics, modes, and frequency analysis types are supported.
- **The design discipline product definition** is concerned with shape representation and assemblies. The geometric shape representations within AP209 are entirely interoperable with those in AP203 that are currently being implemented by most CAD and CAE vendors. There is one additional shape representation unique to AP209 that is utilised to represent the shape of composite constituents.

According to [Hunten, 1997], both design and analysis product definitions may be independently configuration controlled, and many aspects of each are subject to approvals. Another crucial concept is that the shape and analysis information is meant to be implemented to enable bi-directional transfer to enable the feedback of information in design-simulation loops. Figure 91 shows the overall information model of an **analysis_discipline_product_definition** (in red) and underlines the relationships (in purple) with its corresponding **design_discipline_product_definition** (in blue).

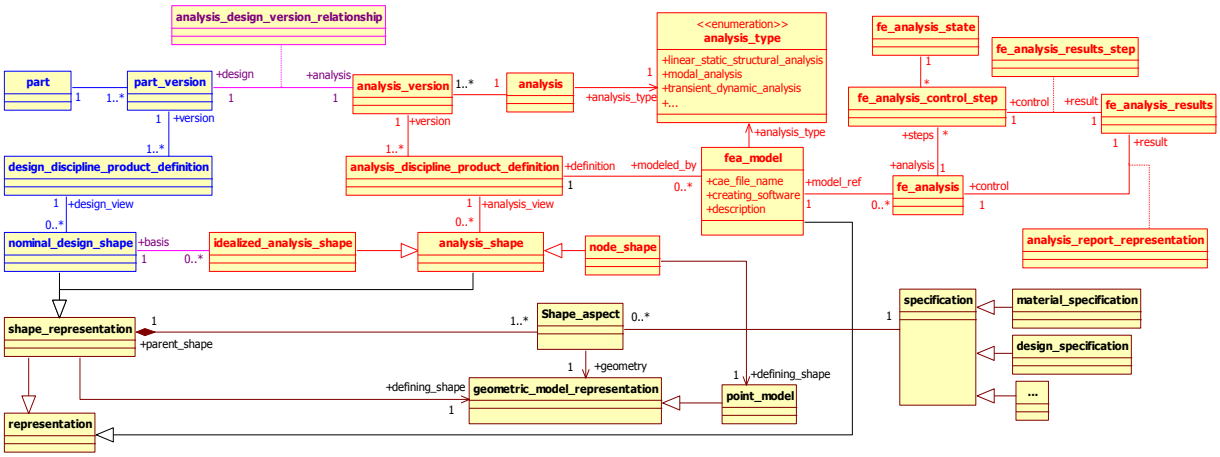


Figure 91: Analysis vs. Design Discipline Product Definition recreated from [ISO, 1999]

An **analysis** is defined in AP 209 in the same manner as STEP products or parts; i.e. all analysis products are associated with a set of **analysis_versions** (i.e. product_definition_formation); themselves referencing a set of **analysis_discipline_product_definitions** (i.e. product_definition) to establish specific analysis stage views of the analysis information. The **analysis_discipline_product_definition** entity establishes many important relationships (such as analysis to analysis shape and finite element models) to aggregate and integrate all the information required to perform the analysis. AP 209 allows for separate versioning of analysis and part (design) versions. The relationships between design parts and the analyses that verify their behaviour are established between the corresponding **product_definition_formations**. The part and analysis versions are hence related through a sub-type of product_definition_formation_relationship: the **analysis_design_version_relationship**.

An **Analysis_shape** is a shape for a **Part_version** as defined for the specific needs of an analysis. An **Analysis_shape** is whether an **Idealized_analysis_shape** or a **Node_shape**. An **Idealized_analysis_shape** represents a shape on which points are associated for the location of nodes in a finite element model, or a shape that is suitable for mesh generation. Mesh generation shape may include the topological information necessary to specify mesh generation curves, areas, or volumes, or may be just the bounding geometry of the mesh generation curves, areas, or volumes. The geometry may be the **Nominal_design_shape** or geometry that has been idealized by generating shape information that is derived from the **Nominal_design_shape**. A **Node_shape** is defined by the points of the nodes in a **Fea_model**. In AP209 the analysis shape is directly associated with the **analysis_discipline_product_definition**. However, to be compliant with other STEP APs (especially AP203) and as shown in Figure 92, the link between the configuration management data for an analysis and its shape is defined with the entities **product_definition_shape** and **shape_definition_representation**.

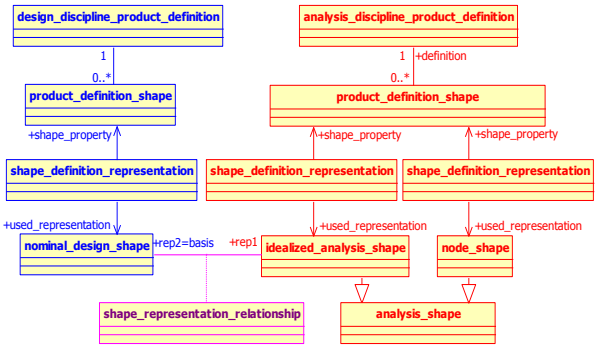


Figure 92: Recommended data structure for relating Analysis Shape to Analysis

As shown on both Figure 91 and Figure 92, AP209 provides the ability to link the **idealized_analysis_shape** to the actual **nominal_design_shape**. The **design_specification** entity shown in Figure 91 can be used to specify intent of the designer in order to create the correct idealized shape. However, it is not clear in the presentation how the idealized shapes are created. AP209 provides a mechanism to relate the shapes, but does not provide any mechanism to capture modelling methodologies regarding the analysis type and objectives.

Research in the development and improvement of the STEP standards to address design-analysis integration issues has continued. In [Gabbert&Wehner, 1998] and based on current step-based CAD-FEA coupling approaches, authors propose the concept of an object-oriented CAD-FEA integration data model to allow for bidirectional and process conform data exchange. The **Engineering Analysis Core Model** (EACM) is part of the STEP standard suite and describes the way that engineering analysis data are stored and the way that engineering analysis information is exchanged. According to Leal, AP203

and 209 are useful but limited in scope [Leal et al., 1999]. The goal of EACM is to increase the scope by capturing all engineering information to support business practices. The EACM deals with three key aspects in engineering information management:

- The management of engineering analysis and design information;
- The linking of engineering information to activities, decisions, and analyses;
- The storage of information about a product in time and space.

In [Charles, 2005, Charles&Eynard, 2005, Charles et al., 2005, Charles et al., 2006], authors investigate the management and integration of CAD and FEA Data in PDM/PLM environments. These works lead to the specifications of a simulation data management environment within a PLM approach. This environment, named EGDS, proposes an innovative modelling of simulation cycles through the adaptation and the enrichment of data management methods of the PLM approach. In order to guarantee its interoperability, EGDS also implements a neutral format dedicated to simulation data management exchanges based on STEP AP209 standard: the **SDM schema**.

In [Nguyen Van, 2006], the author specifies the so-called "**Collaborative schema**" that aims mainly at ensuring simulation and design data interoperability and transactions as well as trans-enterprises exchanges. A part of this step-based data model has been implemented within the EDM framework demonstrator developed within the VIVACE project [Tabaste, 2005] and in collaboration with MSC software. In view to ensure a better consistency of data conveyed between design and simulation departments, this framework has especially demonstrated the capability to perform a standardized data migration between the Snecma PDM system (ENOVIA-VPM from Dassault Systèmes) and the SimManager tool developed by MSC software. These latter mentioned research works are the first implementation and development of Simulation Data Management principles.

More recently in [Gujarathi&Ma, 2010, Gujarathi&Ma, 2011], authors propose a CAD/CAE integration method using a common data model containing all the required parametric information for both CAD modelling and CAE analysis. This data model is used as a parametric data model repository and as the supply source of input for those associative entities of CAD and CAE models and thus maintaining the associative dependences among them. The structure as well as the CAD-CAE data flow is governed according to design-simulation loop processes. According to authors designers can hence relate the expected scenarios with the engineering changes proposed and can take the parametric actions accordingly. CDM acts as the centralized parametric input for computer modelling software tools through their APIs. Their method suggest that the common data model gets modified during each development cycle according to designer's intent, the changes in it are consistently reflected in both CAD and CAE models through regenerations and analysis iterations semi-automatically. However authors do not provide the detailed data structure of this data model and they do not specify neither the way the developed software prototype integrates APIs and interoperate with other domain-specific tools that can populate and/or modify the CAD and CAE models parameters. The method to integrate modelling knowledge rules within the data model and within the engineering procedures managed by the developed software prototype remains ambiguous.

8.1.4 Multi-view and multi-domain aspects in PDM

Several research groups are exploring design-analysis integration through a multi-aspect modelling paradigm. With the advent use of collaborative CAD modelling in the end of the 90's, researchers have brought into focus the requirement of multiple views and representations of the same design

object by different design disciplines. Indeed through the entire lifecycle of a product and through the various disciplines involved in its development, the related engineering data can be viewed through different domain-specific aspects or views by applications to generate, display, or modify the product information. Hence, each designer's view and representation must be customised and integrated within any comprehensive representation of the design under concern.

In [Rosenman&Gero, 1996], authors introduced the concept of product views based on functional contexts. According to authors, a view prescribes the relevant functional sub-systems which in turn prescribe a particular model of a design object, i.e. which design prototypes, design elements and properties are relevant to that view. They argue that the representation of functional properties is the essential aspect of the modelling of multiple representations.

In [Tichkiewitch, 1996, Tichkiewitch&Véron, 1997], authors present an integrated design methodology and product data model enabling the representation of the product through the different views of any participant. The product views are defined through the various required combination of system components, links and relations. A link is a characteristic of a specified component which allows an external consideration of the component. A relation may be specified between two or several links of the same component or between links of different components. The various required combinations are defined by a grammar based on the rules of **decomposition**, **substitution** and **multi-view representation**. Decomposition of a component allows different levels of abstraction to be displayed and specified. Substitution realises the possibility of replacing a relation between several links by a set of components, links and relations, in order to specify how the replaced relation is realised. Contrary to the decomposition, the substitution does not change the level of abstraction that is studied. A three layered CAD design modeller prototype supporting these concepts is presented. The first layer is the management layer for the product database access permitting to filter and select the relevant data required by the user. The second layer includes classical CAD software building blocks such as an inference engine, a geometric development kernel and a feature based engine. The third layer enables the use of external software and encompasses all graphic applications which permit to display the specific representations (e.g. realist 3D visualisation tool or a mesh generator).

In [Yoshioka&Tomiyama, 1997], authors present a mechanism for integrating various aspect models, such as geometric, kinematic and finite element models. The KIEF (Knowledge Intensive Engineering Framework) is constructed using these multiple objects (i.e., aspect models) expressed through a meta-model mechanism. The meta-model represents the relationships between the concepts in the aspect models. The framework hence integrates and maintains the consistency of the various models. The KIEF framework also integrates commercially available software tools through a "Pluggable Metamodel Mechanism" [Yoshioka&Tomiyama, 1997]. An aspect model is a model of a designed artefact from a particular point of view. For example, a FEA aspect model may be completely different from the geometric shape aspect model. Aspect models are built by first constructing relationships between models. The meta-model mechanism provides the framework for integrating the many aspect models associated with a technical artefact. Aspect models are built by determining the level of abstraction desired, determining the appropriate simplification needed, and finally by the exchange of data between aspect models. The meta-model mechanism also describes how information is exchanged among the aspect models. However, it is not always easy to extract all the necessary parameters to complete the aspect model. For this reason, the ability to plug in existing modellers is presented. However, it is not clear how the various modellers share product information to support the various aspect models.

In [De Martino et al., 1998], authors introduced an approach to CAD-CAE integration based on design-by-features and feature recognition. The developed integrated feature-based CAD modeller allows the user to design with features and to derive different context-dependent semantic and feature-

based representations from solid models. The sharing of semantic information across engineering applications and domains has been achieved through the development of an “intermediate model” which is the common shared product data model that maintains a homogeneous, multiple-view, feature-based representation of the part. From this model, specific views can be derived. To maintain consistency between different views, only the designer client is allowed to modify the model. In this system, a single data structure stores the information of all views. This system allows application tools to incorporate application-specific data into the shared intermediate model. Further, it allows applications to extract specific data needed in the various contexts. At this time this work was a major step towards application tool integration based on a common shared and feature-based object model.

In [Hoffman&Joan-Arinyo, 1998] authors introduce a similar approach based on the “**product master model**” paradigm. The design-analysis association is predicated on the master model being an object-oriented repository that has mechanisms for maintaining the integrity and consistency of the information structures for the various engineering domains. Additionally, the master model has several clients, one of which is the CAD application responsible for creating the initial net shape and also for modifying the net shape. For each additional clients associated with the master model (such as CAE applications, tools for manufacturing process planning, for cost estimation, etc.) there is a corresponding view of the product. Each client application can deposit product information it processes to the master model, as well as keep a private repository of information relevant to itself.

In [KwangHoon et al., 2003], uses a combination of a multi-level modelling approach which consist in building a feature-based design model and mapping it to executable representations of secondary “viewpoint models”. This multi-level modelling approach is implemented in three-level architecture. Top of this level is a feature-based description for each viewpoint, comprising a combination of form features and other features such as loads and constraints for analysis. The middle level is an executable representation of the feature model. The bottom of this multi-level modelling is an evaluation of a feature-based CAD model obtained by executable feature representations defined in the middle level. The proposed system has two stages of mapping between models. First, the mappings concerns mapping between the top level feature representations associated with different viewpoints; for instance for the geometric simplification and addition of boundary conditions associated with moving from a design model to an analysis model. Then, the mapping is done between the top level and the middle level representations in which the feature model is transformed into the executable representation.

In [Yan, 2003], the author proposes to model a product or system from multiple perspectives in order to generate a complete virtual model representation of the product to support multi-life phase design decision exploration and decision making.

A similar approach to the one suggested by [De Martino et al., 1998] has been used by [Bronsvoort&Noort, 2004] to develop a multiple-view feature modelling methodology to support conceptual design, assembly design, part detail design and part manufacturing planning. This methodology does not only provide views with form features to model single parts, as previous approaches to multiple-view feature modelling did. It also provides a view with conceptual features to model the product configuration with functional components and interfaces between these components as well as a view with assembly features, to model the connections between components.

In STEP, and as defined in Part 44 [ISO, 2004a], since the effectivity is related to a product_definition_relationship, many different views of the effectivity can be established by varying the relating_product_definition. For instance, effectivity can be maintained based on the "design" product_definition and another based on the "manufacturing" product_definition. The various product_definitions can move into other life cycle stages for the design as well. In this way, effectivities can be defined for any of a number of views and life cycle stages of the design.

AP239 and AP233 information models enable to manage multiple evolving product views and product structures and hence enable the identification of the various systems, functions, or physical parts, together with identifying zones within the product or a combination of these (hybrid view). This is done by defining different types of product breakdowns among which functional, system and physical breakdowns. A breakdown is always related to a design or a real product, of which it is a breakdown. It is identified and versioned as an object in its own rights. It has a number of constituents (breakdown_elements), often structured hierarchically, that makes up the breakdown structure. It is possible to relate a breakdown_element to breakdown_elements in other breakdowns, but a specific breakdown_element will always only be part of one product breakdown. Each breakdown_element may or may not relate to any number of designs (assembly structures) of which one should be used to realize the constituent [ISO, 2004b, ISO, 2012]. Figure 93 captures the essential areas for representing a breakdown.

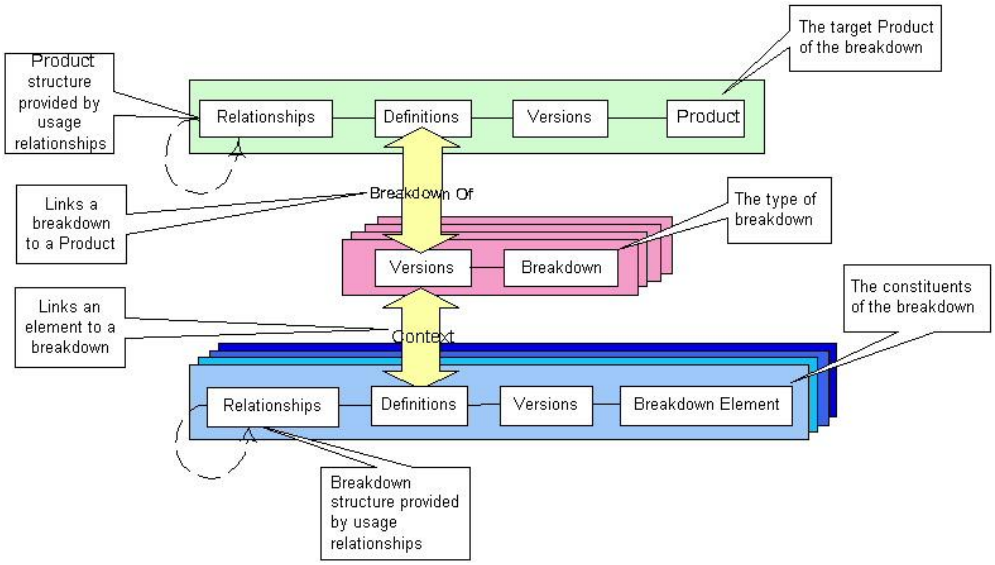


Figure 93: Conceptual working principle of the Breakdown Structure Model as defined in [ISO, 2004b, ISO, 2012]

As shown in Figure 93, breakdowns are represented as a set of objects that may be change managed and version controlled. The breakdown of systems into components can also be change managed and version controlled. The breakdown itself is processed separately from the thing it breaks down. Entities in the green area represent the product under consideration. Those in light-red represent intermediate concepts between the product and its elements (the Breakdown itself). Those entities in shades of blue represent the numerous elements of the product. Those in light-yellow indicate where relationships are used to link the concepts of the different levels. These colours are also displayed in the following UML class diagram of Figure 94 that shows how breakdowns are identified and related to the thing they break down.

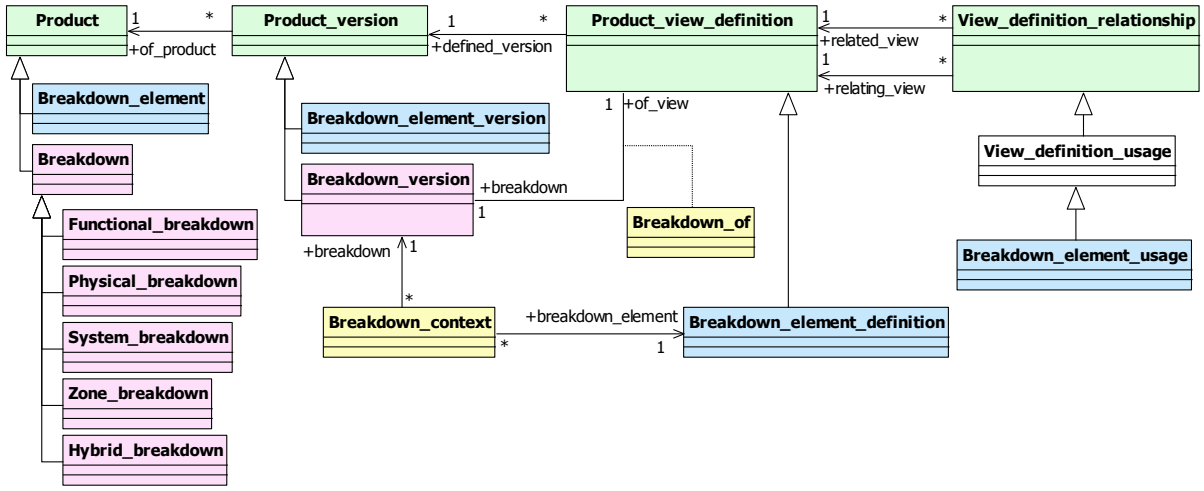


Figure 94: UML class diagram representing breakdowns as defined in [ISO, 2004b, ISO, 2012]

Neither AP233 nor AP239 specify behavioural breakdowns to manage analysis breakdown models and to relate the corresponding behavioural breakdown element (e.g. simulation models) related them to functional, system or physical breakdowns of a product. However, AP239 allows other types of breakdowns to be defined and used through the definition of reference data. An additional work is hence necessary to define these reference data enabling explicit specification of the bi-directional links between functional, structural and behavioural elements of product structure breakdowns; which is required to ensure continuity, consistency and traceability between design and analysis data across the multiple domain-specific views of the product. Configuration management in AP239 has proved to be a particular challenge [Pratt, 2005]. Earlier parts of the STEP standard handle product definition, but AP239 requires knowledge of the permitted, required and actual configuration of every individual product in a product class. Each individual may be modified over time to meet updated requirements. It is hence necessary to generate multiple application views of the data at each life cycle stage, and to manage historical archiving responsible for the generation of feedback.

8.1.5 The multi-view consistency and multi-domain engineering change propagation issues

Previous research efforts on product data views and their consistency maintenance have concentrated mainly on limited and specific areas, such as multi-view geometric features. In [Hoffman&Joan-Arinyo, 1998], authors raise crucial issues regarding the capacity for the product master model to maintain the integrity and consistency of the information structures for the various engineering domains that populate the master model repository:

> *"…the data in the master model originate from different domain-specific programs, how can this information be kept consistent and how is it maintained under design changes? In our view, the CAD system is one of the clients of the master model, with the primary charge of creating and maintaining the net shape information. … How can we establish and maintain a persistent association between the geometry data contributed by the CAD system and data originating from other application programs? "*

As continuation of the work presented in [Hoffman&Joan-Arinyo, 1998], the same authors present three mechanisms for maintaining consistent product views in a distributed product information database [Hoffmann&Joan-Arinyo, 2000]. The mechanisms are used when one of the views makes a

change to the product model and the other views must be updated to maintain consistency. When a change is made to a model (deposited on the master model repository), a protocol is followed to ensure the most up-to-date product data is available to all clients. The change information is posted and it is up to the clients to re-associate with the new information. In the earlier paper, authors identified two mechanisms for maintaining consistency of views, an external information association mechanism, and a constraint reconciliation procedure. In the second paper they expand on the details and applicability of those mechanisms and add a third mechanism as a complementary technique for maintaining consistent views under distributed updates. The master model updates only concerns shape changes, changes of parameters, dimensions, constraints and changes of model attributes. According to the authors, shape changes are the most difficult ones to respond to. However, not all changes can be automatically propagated across the various product views and authors suggest whether imposing restrictions on shape changes whether involving human intervention if required. They demonstrated that it is possible to automate a wide range of view updating operations while preserving privacy of proprietary information. However authors underlined the difficulty for maintaining different feature views with the current history-based CAD design approach [Hoffmann&Joan-Arinyo, 2000]. Their algorithm only partially automates change updates; they tried to apply a set of techniques familiar from the feature recognition literature when dealing with updating the feature history. They found that in many situations an adjustment is possible purely by constraint reconciliation. Were it not for the sequential design history implemented by CAD systems, constraint reconciliation would be more widely applicable. Finally, the maintenance of attributes can be completely automated, and, with it, the maintenance of many downstream views that can be derived from attributes and relations maintained on the net shape elements.

According to [Hoffmann&Joan-Arinyo, 2000], to deal with the multiple-view and multi-applications data consistency and association issues, developed systems are organized as either one-way or multi-way architectures. In one-way architectures, features in an application view are derived from the features that belong to a privileged view, usually the design view. The designer defines this view and conversion modules derive application-dependent feature models. If a modification is required by a downstream application, it must be entered in the privileged view first. Only thereafter can one derive new, application-dependent views [De Martino et al., 1998]. In the one-way approach, feature conversion is triggered whether when the design is considered completed whether incrementally after each feature attachment operation in the design view. In multi-way architectures, modifications required by an application are introduced in the view in which the need for them arises, and each modification, in any view, is propagated automatically to every other view. However very few works explain precisely how a feature in a specific product design view can change the net shape of another related product design view and there is a paucity of techniques to formalize such changes. In [Bronsvoort&Noort, 2004], authors specify the net shape by a cell complex where the cells are refined such that every feature of an application view is composed of entire cells. That allows one to edit shape mechanically in any feature view and to achieve consistency across all views using constraint techniques. If an inconsistency between different views is found, the approach rebuilds the view that generated the inconsistency. The view is rebuilt incrementally by first removing some features and then adding new features.

Modelling approaches for multi-view feature can provide only limited product views and consistency maintenance for a small group of design or manufacturing engineers. Although these researches are relevant for ensuring multi-view consistency between CAD feature-based product views, they do not support company level consistency maintenance between other kind of product data views (such as CAE or simulation-driven product data views).

[Shah et al., 2009] believe that a common language such as SysML can serve as a unifying language between the various views of a system. Authors use SysML for defining the high-level relationships that exist between functional, structural, and behavioural product views and related models.

Their method provides the designer with the ability to trace decisions made to corresponding requirements defined in SysML as well as maintain bidirectional consistency between other domains that are linked with SysML. As shown in opposite Figure 95, the SysML generic model can be represented according to three packages and a parametric diagram enabling the constraint relationships between system views.

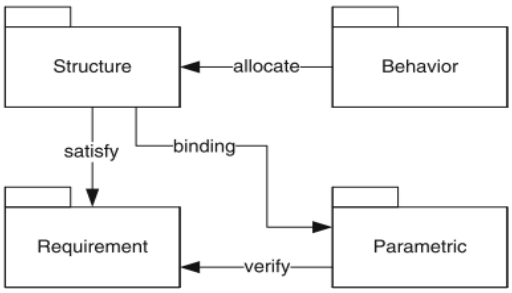
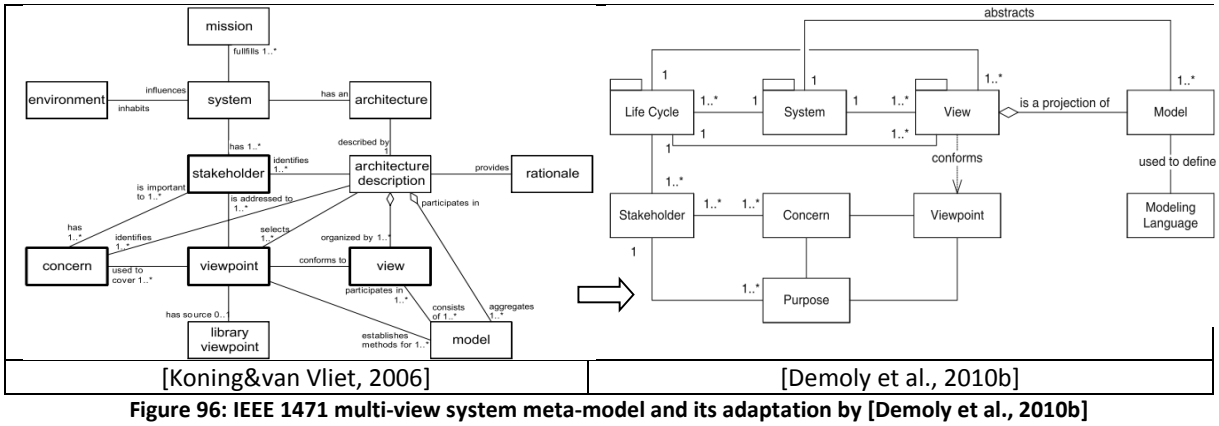


Figure 95: SysML generic meta-model

In [Demoly, 2010, Demoly et al., 2011c], the MUOVA model defines a set of interrelated product views (functional, behavioural, structural, geometric, technological and contextual) according to profiles (role, concern, concepts, business process, etc.) of the stakeholders involved in assembly oriented design issues. The MUVOA model is based on the model proposed by [Gomes&Sagot, 2002] called Multiple Domains and Multiple View-points (MD-MV) which is broken down in domains (project, product, process, and usage) and viewpoints (functional, structural, behavioural, geometric, and physical). As shown in Figure 96, authors propose a meta-model adapted from IEEE 1471 Standard [Koning&van Vliet, 2006] enabling the logical definition and identification of viewpoints/views and domains for the system.



[Koning&van Vliet, 2006] | [Demoly et al., 2010b]

Figure 96: IEEE 1471 multi-view system meta-model and its adaptation by [Demoly et al., 2010b]

The MUVOA model considers activities within the product lifecycle as network of business domains. In such a context, each domain corresponds to a product lifecycle stage and is defined as a system integrating views and viewpoints. Each view represents system with the perspective of a viewpoint. A view-point describes conventions and rules to build and define the related view in order to fulfil stakeholders' concerns. Based on this comprehensive multiple viewpoints model, a novel product relationship management approach entitled PROMA has been proposed. Authors underline the importance of having a full representation of these relationships to facilitate and propagate information flow towards other related views as well.

Figure 97 extracted from [Demoly et al., 2011c], shows an example of product views dependences as defined between the generic product engineering domain and the assembly sequence domain. Authors also suggest using SysML Package diagrams to organise identified domains and related views for the proposed MUVOA model.

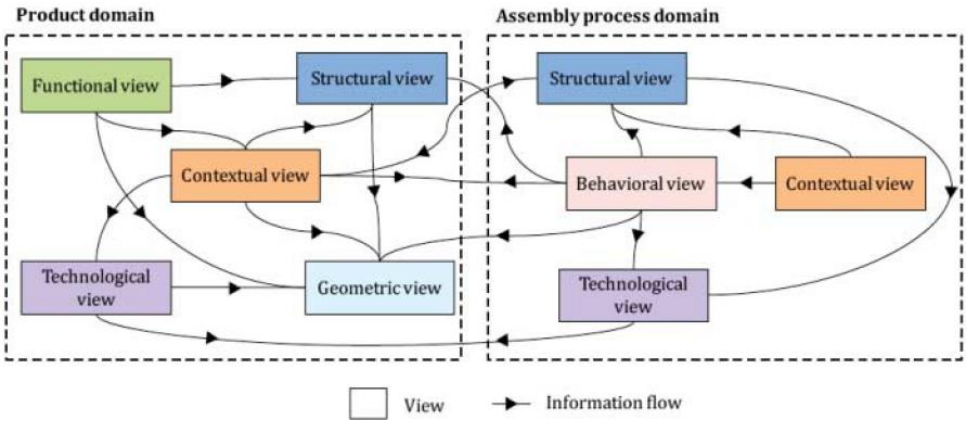


Figure 97: Description of information flow between views defined in the MUVOA model from [Demoly et al., 2011c]

Change propagation is defined by [Giffin et al., 2009] as: "the process by which a change to one part or element of an existing system configuration or design results in one or more additional changes to the system, when those changes would not have otherwise been required".

As defined in the PDM schema, most STEP APs only provides data structures for representation of the data used to manage the work being done during an engineering release and/or change process. The work management area contains the constructs to describe initial part design requirements and the change requirements and issues for revising part designs, as well as the proposed work and the directive for work to proceed in the development of these initial or modified part designs [ProSTEP_iViP, 2002]. However they do not provide the required data structure and mechanism to capture the detailed description an engineering change object. An engineering change object must capture both meta-data about the design artefact (e.g. derivation link between the previous and the new item_version or product_defintion) that has been changed but also the detailed technical data that have been changed (e.g. referencing the geometric feature/parameter or physical properties (e.g. material property) that has changed and that is responsible of the new product_definition release). The AP214, is the only AP that provide a data structure for defining an engineering change object.

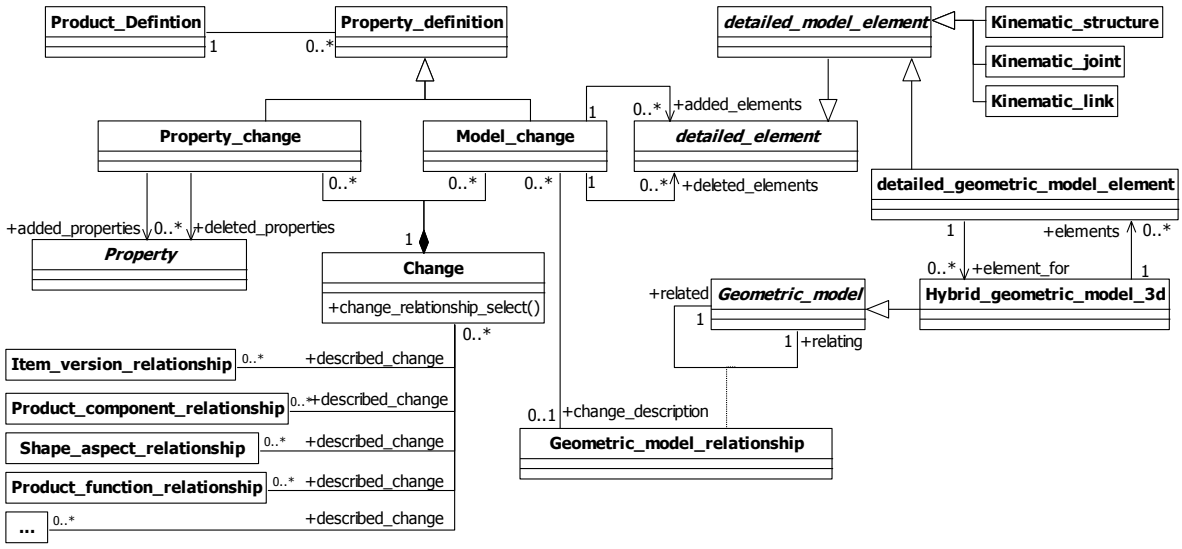


Figure 98: UML class diagram for managing engineering change in AP214

As shown in Figure 98, a **Change** in AP214 is a composition of a set of **Property_changes** and/or **Model_changes**. **Property_changes** and **Model_changes** entities provide the mechanism required to describe the differences between two objects concerning the properties and/or the models describing these objects. A change hence capture the differences between the two objects referenced by the

described_change attribute which can reference different kind object relationships such as **item_version_relationship**, **product_component_relationship**, **shape_aspect_relationship**, etc.

In [Bettaieb&Noel, 2006], authors propose an integrative approach for the definition of a generic framework to schedule domain-specific business models synchronisation. In this approach, a common model is shared by every designer. Analysts' results must be synchronised before changing the common model. Authors describe various options for the synchronization process and propose a generic framework providing every synchronisation schemes. The main ideas are to anticipate the change notification towards interested designers without changing the shared model and to assist the selection of the right time when designer results must be synchronised.

In [Do et al., 2008], authors propose a procedure for engineering change propagation in order to maintain consistency between various product data views. This procedure is based on a product data model which integrates base product definitions for product design, and product data views for other manufacturing or customer support. The product data view in the proposed model enables manufacturing or customer support engineers to define their own product data views, without copying the existing product definition. In the proposed data model, the engineering changes provide structure-oriented change history, effectivity management for production, and integration with product configurations. Based on the integrated product data model, the proposed procedure propagates engineering changes to product data views using the history of product structure changes. However, since the proposed propagation procedure is based on a non-standardized product data model, its application is limited.

Matrix-based approaches such as Design Structure Matrix (DSM) have been largely used for the quantitative investigation of change propagation. For instance, in [Clarkson et al., 2004, Eckert et al., 2006], the Change Prediction Model (CPM) uses the DSM representation of a product to trace potential propagation paths among its interconnected components. Similarly, [Giffin et al., 2009] extend the DSM concept to create the Change DSM for identifying instances of change propagation from one component to another by mapping and documenting interconnections between subsystem areas including physical connections, as well as information and energy flows. Authors also use Graph theory and directed acyclic graphs to draw Change networks. In these graphs, the nodes represent the change requests and the edges the change request dependencies that have emerged during a development program. These change networks are then broken down into one-, two-, and three-node motifs as the fundamental building blocks of change activity. A change "motif" is a simple pattern of connected change requests from which more complex change networks emerge. The idea of the change motifs concept is to provide a means of systematically analyzing change networks as well as the definition of three indices to quantify each sub-system area in terms of its propensity for accepting, reflecting or propagating changes (change propagation index).

In [Ahmad, 2010], the author presents a framework to create a model which captures the four domains of requirements, functions, components/subsystems and detailed design process and subsequently shows how the resulting models could be used to generate "cross-domain models". Elements are linked within and across these four domains via a systematic approach, which allows representation of key aspects of designers' knowledge regarding the change process. The dissertation shows how changes in requirements can be viewed as propagating through these four domains to cause rework in the design process, and how a traceability approach based on the data model can be used to help reason about the cost of implementing a given requirement change.

In [Hamraz et al., 2012], authors propose a multi-domain model which combines concepts from the FBS structure model from Gero and the change prediction method from [Clarkson et al., 2004,

Eckert et al., 2006]. The proposed FBS linkage model is represented in a network and a corresponding multi-domain matrix of structural, behavioural, and functional elements and their links. Change propagation is described as spread in that network using principles of graph theory. Authors have demonstrated that the FBS linkage model accounts explicitly for all possible dependencies between product elements and allows capturing and modelling relevant change requests. It also provides information of why and how changes propagate and the model is scalable to different levels of decomposition and levels of abstraction.

In [Pasqual&Weck, 2012], authors introduce a multilayer network model (see Figure 99) integrating three coupled layers, that contribute to change propagation: the product, the change and the social layer. The approach place engineers in the social layer. They work on changes in the change layer that affect components in the product layer. The multilayer network model captures the interactions within and across the product, change and social layer. Each layer of the multilayer network model consists of a distinct, directed network composed of nodes connected by intra-layer edges. The product layer is a network representation of the product or system being designed. The nodes of the network represent product components and associated documentation (e.g. requirements). The product intra-layer edges of the network represent technical interfaces among the components. The interfaces can be physical connections or channels for the flow of energy or information. The change layer is a network representation of change propagation. The nodes of the network represent individual changes or change requests. The change intra-layer edges of the network represent propagation relationships among the changes. As in [Giffin et al., 2009], directed edges can identify parent-child relationships, while bi-directional edges can identify sibling relationships between children of the same parent, or two changes related in a significant way. The social-layer is a network representation of the organization. The nodes of the network represent teams, sub-teams, or individual employees. The intra-layer edges of the network represent various relationships among individuals and groups. The other half of the multilayer network model consists of the inter-layer edges represents the critical relationships between the layers of the model (to relate a change to a person or to a product component).

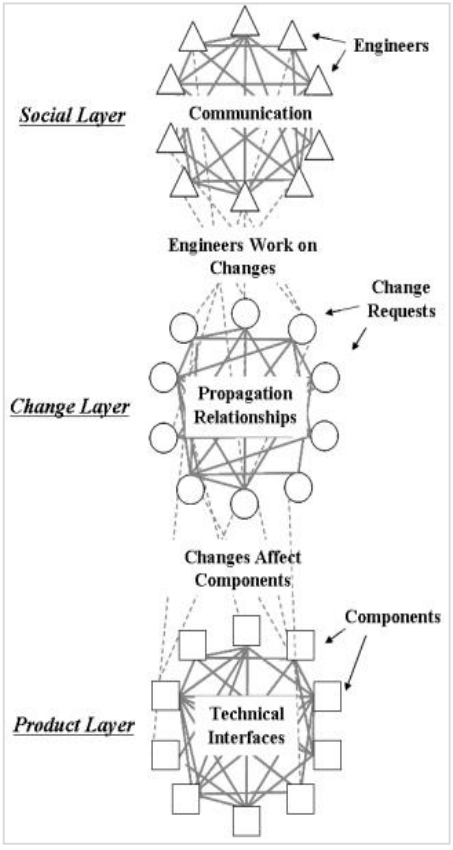


Figure 99: Multilayer network model from [Pasqual&Weck, 2012]

These matrix-based and graph-based approaches are relevant to analyze a change propagation network and hence predict the potential change impact chain that will occur in future development programs. However they do not specify the data structure that is required to implement in PDM systems in order to capture and notify the engineering changes, their detailed description and their potential impacts on other design artefacts definition. Engineering Change data models also have to dynamically propagate these changes across the potential multiple domain-specific and multi-level product views that are concerned by this change.

8.2 Conclusion on current PDM and SDM remaining gaps

This chapter has permitted to review exhaustively various standardized or non-standardized product data models and other approaches for product data structuring and for re-use of product data among various applications.

The use of an MBSE approach supporting digital integration chains challenges introduced in chapter 4 and especially the design-analysis integration requirements have lead to consider several aspects of product data models that are necessary to implement in PDM systems. These data models have been analysed in detail in order to identify the relevant data structures that are necessary to provide and support a flexible and multi-view DMU environment that can become the common referential framework for the design and analysis data used within digital integration chains.

In addition to existing and already implemented product data management capabilities (such as product definition and versioning of this definition, management of part properties, management of CAD data and other technical documents, configuration management including management of bill of materials and engineering change objects) other product data structuring functionalities are required to answer the requirements of such an integrated DMU-based environment for digital integration chains:

- Product assembly and interface data models (such as the OAM or the MUOVA model, AP233 and Sellgren interface data models):
- Design-analysis integration data models (such as AP209 or the Charles' SDM schema)
- Multi-view data models (such as AP239): enabling to manage multiple evolving product views and product structures and hence enable the change propagation across these multi-level and multi-disciplinary product views and related system breakdowns.

Concerning the management of interfaces data, current PDM systems only provide information about product-component relationships referring to their functional role through functional item relationships and configuration effectivities. The information about components interfaces (how components are connected together) within the scope of given functions is still missing in PDM data models and hence cannot be managed in DMUs neither as they are supported by PDM systems.

All identified design-analysis integration data models are step-based (AP209) product data models. In these models the entities permitting to establish the relationships between a design_discipline_product_definition and an analysis_discipline_product_definition only enable to manage design-analysis data associativity and traceability at a high level of abstraction (product meta-data level). The relationships at a finer level of abstraction (topological level) are missing. Moreover the **analysis discipline product definition,** as defined in AP209, and its related configuration management rules are not linked to any simulation context definition (such as studied breakdown level, simulation objectives, operational state, requirements to satisfy, etc.).

Concerning the management of multiple evolving product views within PDM systems, several approach have been identified. On one hand, approaches based on the specification of data structures for a product model do not incorporate the mechanisms of shape changes between product views. Usually, the research work focusing on global product model definition has been addressing the specification of the model from top-down, leading to high level data structure rather than detailed studies of the interaction between product technical parameters and its shape. On the other hand, "CAD Multi-view feature modelling" approaches can only provide limited product views and consistency maintenance for a small group of design or manufacturing engineers. These kinds of approach do not support company level consistency maintenance between other kind of product data views (such as CAE or

simulation-driven product data views). Therefore we are convinced that a combination of both macroscopic approaches (defining the necessary data structure to implement in PDM applications to manage design-analysis data integration issues as well as to manage multiple-product data views and the change propagation procedures across these views) and microscopic approaches (defining the required mechanisms for simulation-driven structural and shapes DMU adaptations) is required.

The use of product data standards in PDM and CAX systems is especially important in collaborative and distributed design. It is essential when trying to implement a MBSE approach to ensure interoperability between systems and hence permitting to share/re-use design artefact models and other related product data while avoiding the loss of data consistency when exchanging information between heterogeneous applications.

Collaboration around CAD individual models and assemblies has become a widespread practice in the industry and has been widely used for almost 15 years. Whereas there exist neutral formats (IGES, STEP) and direct translators widely accessible (directly in the CAD packages or as standalone solutions), the simulation side lacks this kind of tools. It is arguable that the data model for describing a finite element simulation is far more complex than for describing a geometric model. However, all the effort put into the creation of the STEP application protocol AP209 and all its underlying integrated resources has not reached the market applications, as AP203/214 did for CAD data exchange. Only AP209 translators prototypes have been developed within some research projects (e.g. in PATRAN from MSC software) in order to ensure interoperability between CAE applications themselves and between CAD and CAE systems.

The issue of formalizing product metadata from an unambiguous and generic point of view is often not addressed by the PDM and/or SDM systems. Indeed, the data that need to be managed within these environments are often defined and structured following the PDM or SDM editor formalism. This often affects the understanding of meta-data and their reuse in other applications. The PDM schema, the AP239 or AP233 but as well NIST's CPM and related extensions have been developed in order to support this interoperability between PDM systems mainly managing product data and meta-data exchange. However no current commercial application is using these standards and information related to simulation activities are not considered in these standards.

Moreover, the interoperability between CAD and PDM systems is currently ensured by PDM and CAD applications developed by the same editor and enabling to import/export as well as synchronizing product assembly structures used in both PDM and CAD systems. We did not identify any study defining a neutral format for enhancing interoperability between data managed in CAE applications (FE models, nodes, models connectivity, loads and boundary conditions, analysis results) and the product meta-data managed within PDM systems. Some companies have developed their own tools to ensure interoperability between FEA data and product and process meta-data. However, no standard format has emerged and the high level data formalism managed by these tools is lost.

These interoperability issues emphasize the need for providing collaborative digital platforms (where both design and simulation data are managed within a common federated environment) supported by a neutral and standardized product data model. Regarding the gaps identified in existing product data models, we believe that this product data model must be based on a combination of geometric and configuration management data standards (e.g. AP203/214/209 and the PDM schema) but also of product lifecycle and system engineering standards (e.g. AP233/239, OAM).

Next chapter is dedicated to the formalisation of our contribution regarding the gaps previously discussed. A "Design-Analysis Integration System Framework" is proposed in order to support a flexible and multi-view DMU environment that can be used as the common referential framework for the design and analysis data used within collaborative digital integration chains.

PART III: PROPOSAL FOR A DESIGN-ANALYSIS INTEGRATION FRAMEWORK

An observation in industrial environment has allowed us to understand and model actual integration chains. This observation phase lasted about six months and was performed through audit and interviews conducted within four inter-dependant design offices and involving ten operational engineers.

For instance, below Figure 100 provides an illustration of the current (As-Is) situation highlighting the various product information flows that occur through a mechanical IPPS digital integration chain.

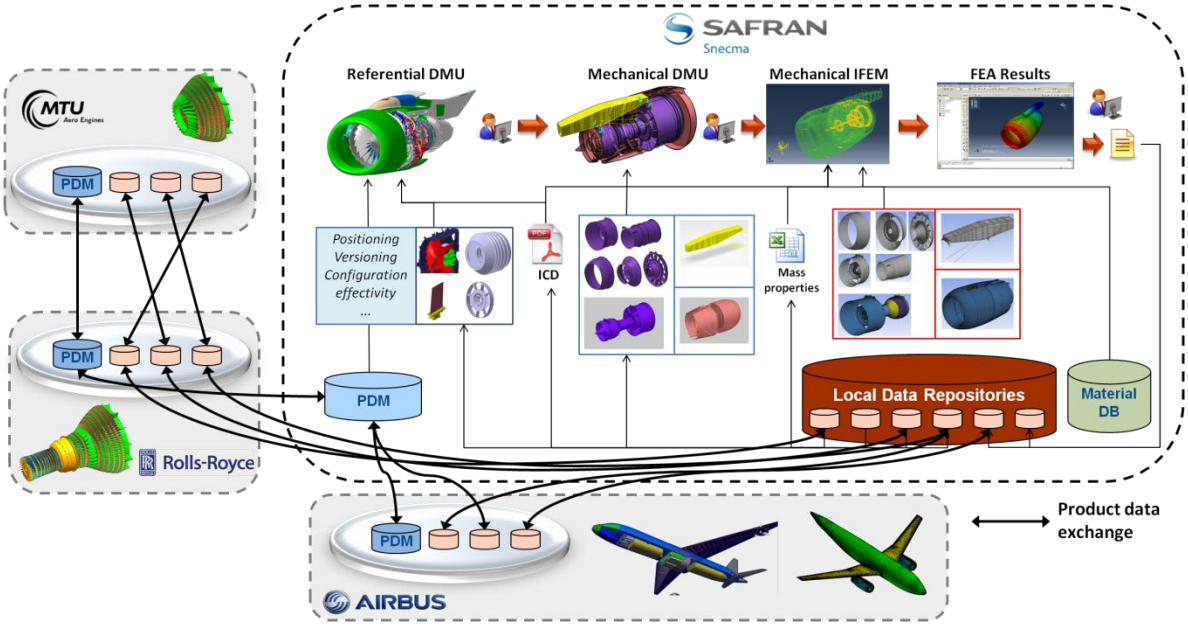


Figure 100: As-Is collaborative mechanical digital integration chain

As we can see in this figure, the assembly structures, the CAD models and their position as well as the configuration data are extracted from the PDM data base to create the referential DMU in the CAD environment. However, when creating this referential DMU or the derived simplified mechanical DMU used for producing the mechanical IFEM, many other data (like Interface Control Drawings (ICD), mass properties, idealised CAD models) are used and spread across various local data repositories. When creating the mechanical IFEM, the FE models but also the mass and material properties are stored and extracted from local data repositories or from dedicated data bases (e.g. the material data base storing all the material references and their related properties). The same occurs when dealing with the FEA results distribution process where the results at the interfaces of the studied system are sent through spread sheets to the various co-designers or analysts concerned by these results. As these data are not managed within a federated environment and are spread across these local data repositories or data bases, it is very difficult to integrate efficiently all these data. This integration must ensure the traceability between design and analysis data but also the consistency between data/models used at various levels of abstraction and for different disciplines in order to enhance re-use and avoid re-work or usage of unsuitable data.

Figure 100 also displays the various and numerous product data exchanges between partners involved in the digital integration chain. Standardized data exchange is only achieved in the area of CAD

data exchange. Most of other product data exchanges between partners (using for instance heterogeneous PDM) are performed through the use of translators that ensure the transformation of system data output into another. This method is not efficient because it requires a lot of different translators due to the variety of tools used in the aeronautical extended enterprise (N^2-N translators if we consider the collaboration between N partners each one using a different PDM system).

The analyses and the reflexions conducted with Snecma experts and other industrial partners has allowed us to propose a picture of the "To-Be" situation as illustrated in Figure 101 below.

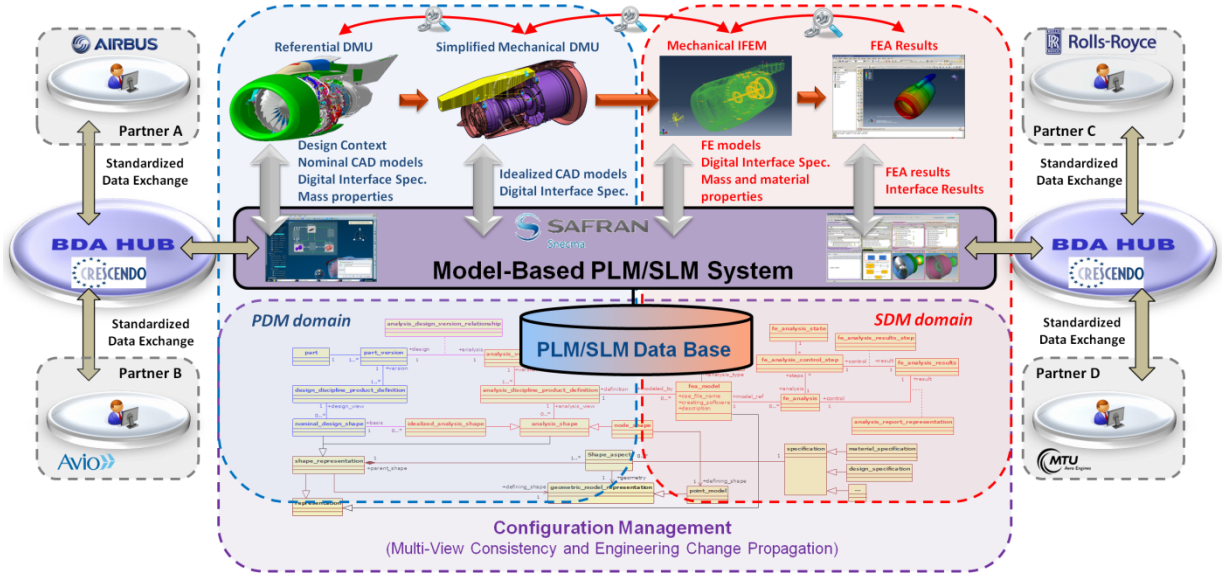


Figure 101: To-Be collaborative mechanical digital integration chain

In order to achieve this situation, the objective of this PhD study is to define the concepts, methods and tools that are needed to provide the federated design and integration environment supporting a flexible and multi-view DMU environment that can become the common referential framework for the design and analysis data used within collaborative digital integration chains. We propose to contribute by:

- Defining a **dedicated integrator environment** based on MBSE concepts to implement in PLM environments in order to:
  - **Specify system architectures** (system's constituents and interfaces) from multiple viewpoints such as different disciplines, life-cycle phases, or levels of details;
  - **Enrich DMU** models with required information (physical properties, interfaces identification);
  - Facilitate the **acquisition**, the **restructuration** and **reuse** of the **appropriate product definition data** to use **for the simulation;**
  - Manage **multiple levels of detail** and **data consistency** between **multi-level** and **multi-domain** system representations;
  - Innovatively **define and specify components' interactions and interfaces** to ensure more efficient contextual design and integration through the use of 3D-based CAD and FE interface templates.
- Defining the supporting **product data model** (mainly based on existing or adapted standards) that will:

- Ensure consistent, seamless generation of design artefacts within an Integrated environment where **functional, logical, physical and behavioural architectures** and related data can be easily defined and consistently integrated;
- Provide appropriate **product definitions views** according to the application domain;
- Manage **digital interfaces objects** and enrich physical and behavioural assembly definitions;
- Consistently integrate **CAD** and **CAE** data to ensure a **bi-directional traceability** between nominal CAD models, idealised CAD models, CAE models and CAE results
- Permit to capture modelling methodologies to perform the appropriate **DMU transformations** regarding the nature of the simulation models and the simulation objectives;
- Ensure **change propagation** across multi-level and multi-domain product engineering views.

Therefore the contribution of this PhD can be summarized as follows:

- Providing the definition of the dedicated “integrator environment” in order to prove the application of MBSE system modelling concepts to mechanical product design;
- Providing the product data model supporting such a framework;
- Identify implementation difficulties by assessing the maturity of existing commercial PLM/SLM applications to implement these concepts;
- Contributing to the definition of the BDA Business Object Model to support our concepts, methods and tools and integrate them in the BDA environment and enhance standardized product data exchange in the aeronautics extended enterprise.

This part of the dissertation is made of three chapters. Chapter 9 describes our research context and details the successive steps of our research and development methodology that led to the development of the concepts presented in Chapter 10. These concepts are related to the contributions above mentioned. Finally, chapter 11 provides a documented conceptual data model (UML class diagrams) describing some of the concepts introduced in chapter 10. In order to provide the DASIF conceptual environment introduced in section 10.1.4, we propose multi-layered data model architecture to implement in PDM and SDM data bases or repositories.

Chapter 9: Research & Development Methodology

9.1 Research context

This PhD has been undertaken within the SNECMA company (SAFRAN group) and more precisely within the Power Plant System (PPS) Integration Division. It has also been carried out within a European research project: the CRESCENDO project. CRESCENDO means **C**ollaborative & **R**obust **E**ngineering using **S**imulation **C**apability **E**nabling **N**ext **D**esign **O**ptimization. This European consortium involves 59 partners representing a cross section of European aeronautics. The project aims at delivering the modelling and simulation backbone of the aeronautical extended enterprise: the **Behavioural Digital Aircraft** (BDA). The BDA concept might consist in a collaborative data exchange/sharing platform for design-simulation processes and models throughout the development life cycle of aeronautics products. However, it is not expected that the BDA is a unique design environment, replacing existing ones. **Instead, the BDA has been considered as a standard data model enabling interoperability, to which existing local design environments and new services to be developed could plug.**

In that context, we have developed a research and development methodology structured around these two complementary research contexts (see Figure 102):

- Within Snecma PPS integration division: we have established a Lean-6σ methodology (explained in next section) in order to clearly analyse the processes, identify the root causes of waste and propose the necessary improvements to make the proposed concepts and capabilities match with the business requirements. Within this division we have analysed the DMU building process, the configuration management process and 3 kinds of PPS FEA process. This methodology has lead to the concepts and capabilities proposal presented in Chapter 10 and the multi-aspect product data model presented in Chapter 11. A first prototype of the proposed design-analysis integration framework has been developed and is presented in Chapter 12.
- Within CRESENDO project: we have been involved in the “Detailed Model Set-Up” work-package. In this work-package, and in collaboration with aeronautical industrial partners, we have defined the PLM/SLM and modelling capabilities requirements to implement in local design environments as well as the necessary BDA data structures to support the related proposed concepts. Within this work-package we have also defined a new methodology to handle digital integration chains and its related scenario test case: the Product Integration scenario. Further, we have monitored the development of the implementation of our methodology and concepts proposal within commercial applications. All these results are presented in Chapter 13.

Both contexts are clearly related. First the concepts and related data models, developed and derived from the processes/tools analyses performed at Snecma, have been standardized and implemented in some packages of the BDA meta-data model. Secondly, the results of the prototypes developed within CRESCENDO have served to assess the maturity of existing PLM/SLM commercial applications to implement our concepts, and to provide the preliminary specifications for the development of a SLM platform at Snecma.

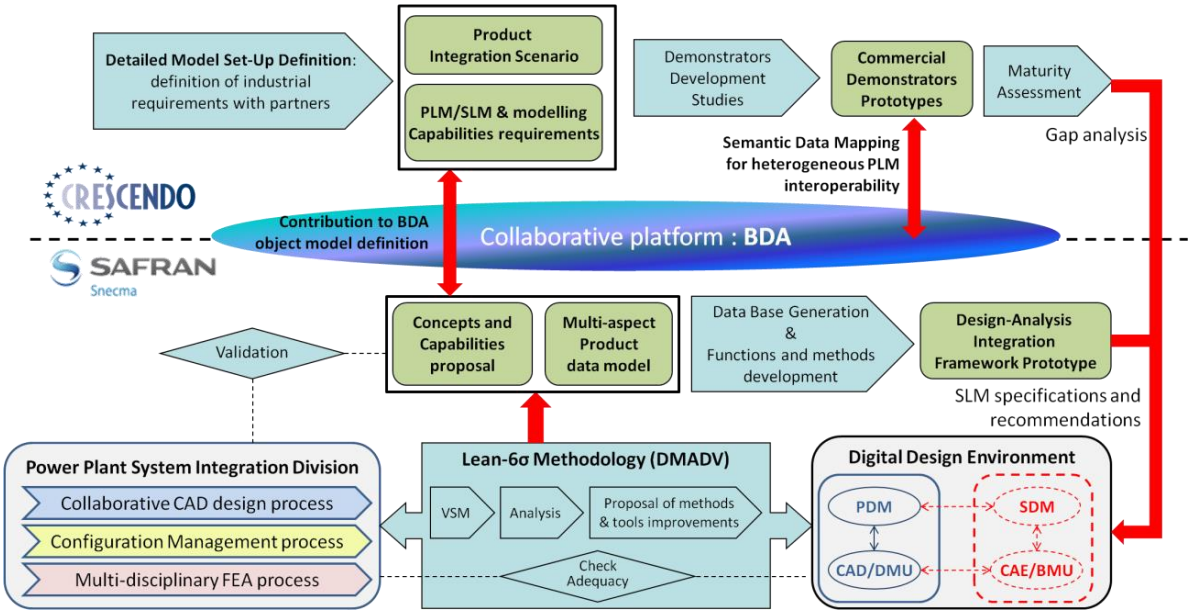


Figure 102: PhD research context and methodology

9.2 A Lean-6σ approach to integrate People, Processes and Tools

In their book, "The Toyota Product Development System", Morgan & Liker define 13 components supporting managers in Lean Product Development (LPD) system implementation (see Figure 103 below).

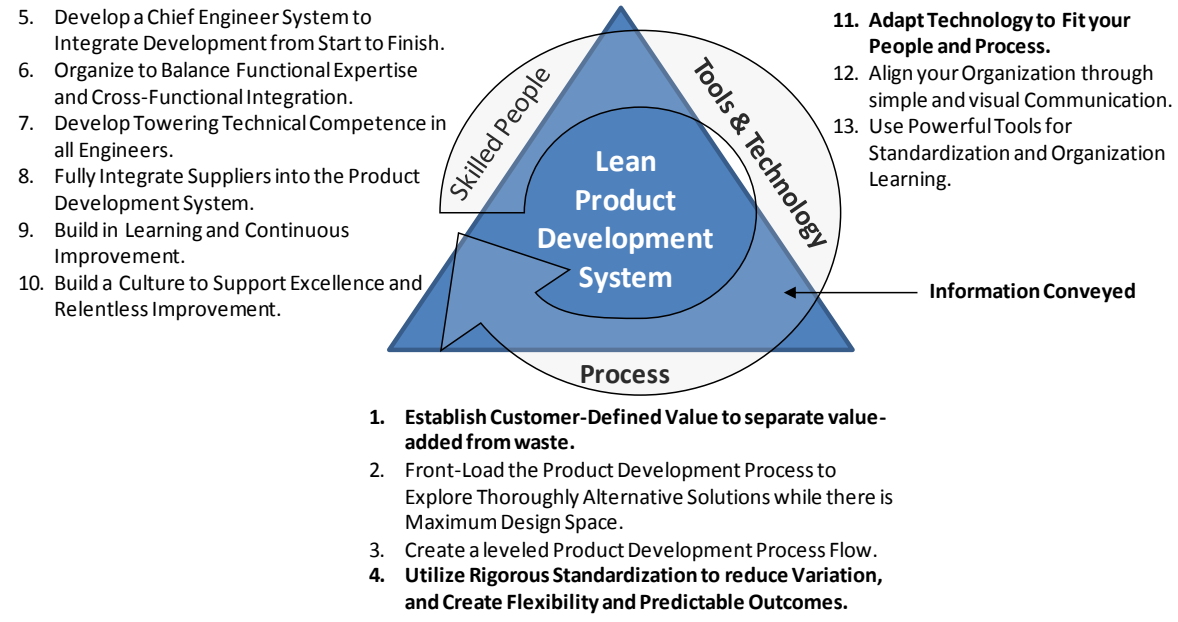


Figure 103: The 13 LPD components (extracted and adjusted from [Morgan&Liker, 2006])

Regarding this system and its components, the major high level challenge for implementing a LPD system is to be able to integrate the triptych: People – Process – Tools & Technology. We propose to add to this triptych an additional transversal component which is the “Information conveyed” through this triptych. Analyzing and optimizing the way information flows through this triptych will contribute to build a learning organization and to capitalize the knowledge produced during PDP and which is profitable for operational value streams. Reducing the time to perform design-analysis iterations requires adopting a lean approach tracking and eliminating waste within these design-simulation loops. With the development, spread and diversification of digital engineering tools used in design-simulation

loops activities (see section chapter 3), we propose to study how to align the triptych Processes - Tools – People and understanding how the information is conveyed through this triptych. This matching can be achieved only considering the entire mentioned triptych, as well as considering the other important LPD component: the human factor (Skilled People). To ensure the scientific validation of the approach, the methodology is based on a Lean-6σ approach (see Table 2): the DMADV (Define-Measure-Analyze-Design-Verify). This approach is based on the methodology described by both [Jugulum&Samuel, 2010] in "Design for Lean Six Sigma" and [McCarty et al., 2005] in "The six sigma black belt hand book". The "Define" and "Measure" phases are dedicated to the identification of these waste and value drivers within the studied process. The "Analyse" phases aim at finding the root causes of the identified waste. The "Design" phase aim at finding the appropriate solutions that will permit to reduce or eliminate these wastes. Finally the "Verify" phase will permit to verify that the implemented solutions are in adequacy with the business processes and users' requirements and that the potential changes and/or rules brought to the process are maintained and respected.

	Process	Tools	Users
Define	the "As-Is" process and identify waste and bottlenecks within the processes	• The current capabilities exploitable by users in the tools • the missing capabilities • the content of the data bases as well as the current information/data model behind	• Users' daily life issues • Requirements and expectations • Users' skills adequacy with digital tools capabilities.
Measure	process performance and metrics enabling to identify waste's root causes	The impacts of missing capabilities or missing information in the data bases as well as the impact of redundant information.	• Users' performance • Impacts of waste related to users' unfitness to exploit tools capabilities • Impacts of users' motivation and frustration due to their daily life issues.
Analyze	the problems impacts and identify the main root causes to address	The impacts of these problems and identify the most problematic issues to address to identify the most relevant areas of improvement.	The root causes of users' daily life issues and identify the most relevant issues to address.
Design	if needed (if wastes are due to organisation of activities within the process) we can propose improvements by proposing a "To-Be" process consistently with the proposed digital capabilities developed in the tools.	• new tools capabilities and/or services based on innovative concepts • the required information/ data model • the required data-base for the demonstrator of the proposed concepts	• Tools' user guides of developed capabilities • Trainings or/and tutorials
Verify	that the new "To-Be" process is more efficient using the measurement system previously defined	• The tools' capabilities adequacy with the business processes and users' requirements • The benefits brought by these capabilities on process performance	• User satisfaction • Ergonomics of GUI • Users' performance • That users are well informed and trained to new capabilities

Table 2: Lean-6sigma framework to integrate Processes, Tools and People

The framework presented above can be managed in several ways. Usually, DMADV projects might last in average six month and generate solutions to the studied problem involving certain rupture and

change with traditional ways of working. The resulting change and potential benefits are permanently controlled with rigorous measures that have been set-up during the verify phase. However, if resulted benefits are not as expected or considered potentially improvable and if time, financial and human resources can be dedicated to it, the approach and the project can be iterative and the sequential DMADV process can become an iterative cycle of continuous improvement.

9.3 Research study

In order to tackle the issue of optimizing the performances of the triptych and to make the expected digital capabilities generic to be exploitable by different working teams having different processes, an experimental study in the aircraft industry is proposed (Figure 104).This study aims at demonstrating how the use of digital engineering and the improvement of existing tools capabilities can contribute to optimize the design and integration processes (including simulation) of the Integrated Power Plant System (IPPS).

The left part of Figure 104 presents the scope of the study encompassing two design teams (mechanical and aerothermal integration), and the triptych Business Processes – Information conveyed – Supporting tools. The potential future tools are related to tools which are being implemented (CAD, CAE and PDM systems) and tools which have to be developed in the frame of the study (the Design-Analysis Integration Framework and the SDM system). The study emphasizes on the use of these tools for acquiring and exploiting data inputs for activities related to mechanical and aerothermal simulations. This study has been identified as it shows difficulties of design related to these two major disciplines in the development of an IPPS. This allow also tackling multi-disciplinary aspects in collaborative product development.

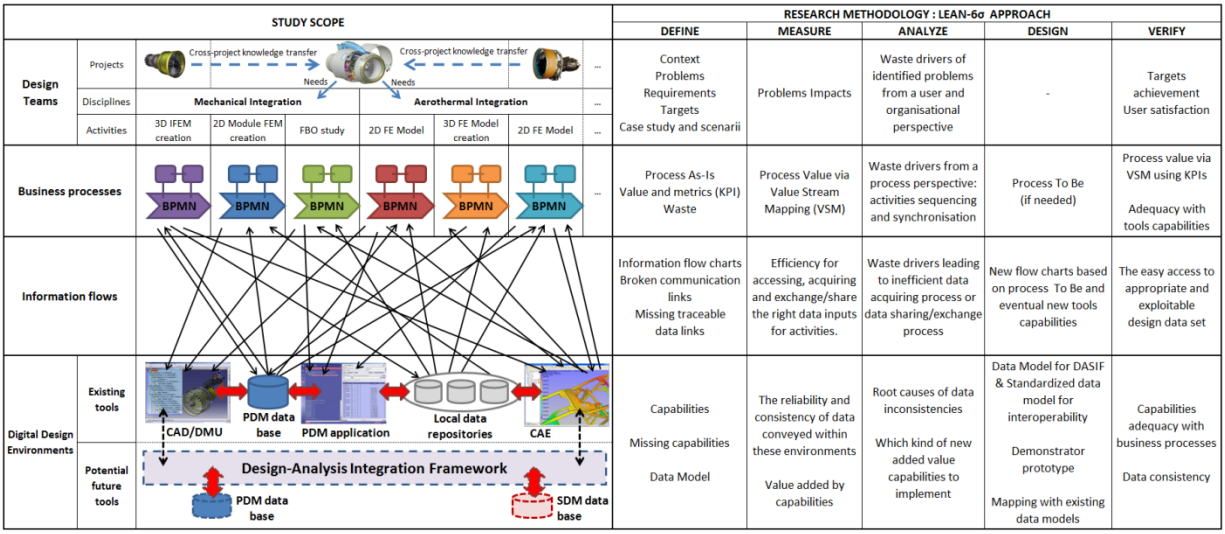


Figure 104: Study scope and Lean-6σ research & development methodology proposal

9.4 "Define", "Measure" and "Analyze" phases

9.4.1 Define phase

At the time this PhD was launched, the Snecma PPS integration division was composed of seven business units as illustrated in Figure 105.

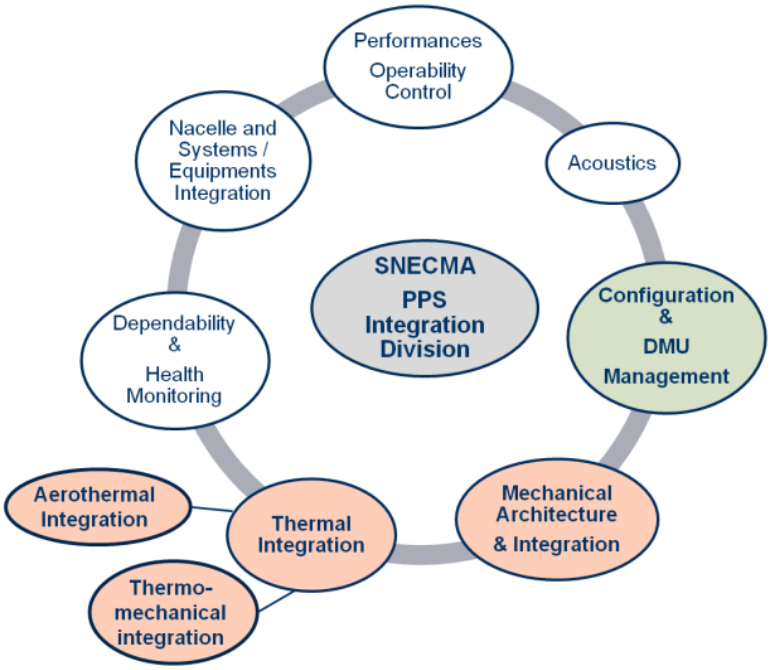


Figure 105: Business units of the Snecma PPS integration division

We have focused our analysis on three inter-dependant business units and related business processes as illustrated in Figure 106:

- the DMU building process and the configuration management process (handled by the Configuration & DMU Management unit);
- and three IFEM creation processes:
  - the creation of a mechanical IFEM;
  - the creation of thermo-mechanical IFEM;
  - the creation of an Aerothermal IFEM.

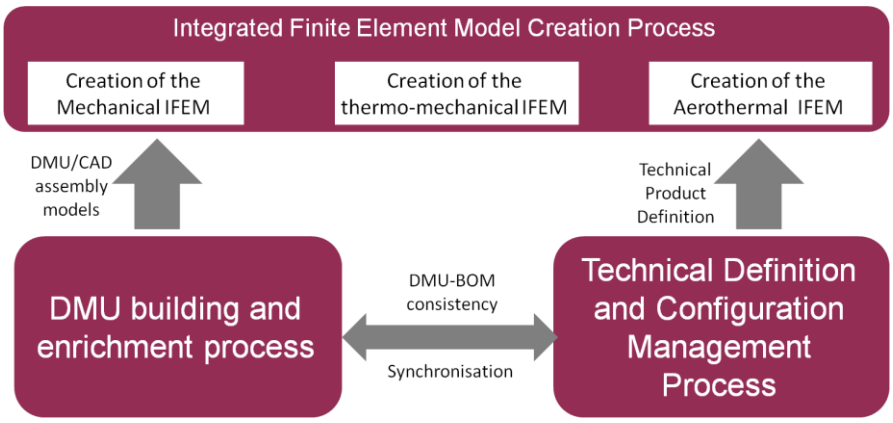


Figure 106: addressed Snecma business processes

The define phase consists in capturing and modelling the business processes mentioned in Figure 106. To provide a detailed map of these processes and their interactions we have performed several "Value Stream Mapping" (VSM). These VSM have been performed and validated in the frame of a Lean-6σ project and have permitted to collaboratively identify the waste drivers and the non-added value steps in these processes. These processes have been modelled in BPMN. These BPMN models capture:

- The sequencing of activities
- The input and output of each activity

- The problems / waste potentially identified related whether to the way of performing some activities whether to the data used for these activities or to some missing capabilities of the digital tools used to perform the activities.
- Value/waste indicators (or key performance indicators) in order to assess the negative or positive impacts of each activity on the global process efficiency and on the value created by this process.

Figure 107 shows an example of a VSM performed for this study. Unfortunately this picture has been blurred for confidential reasons.

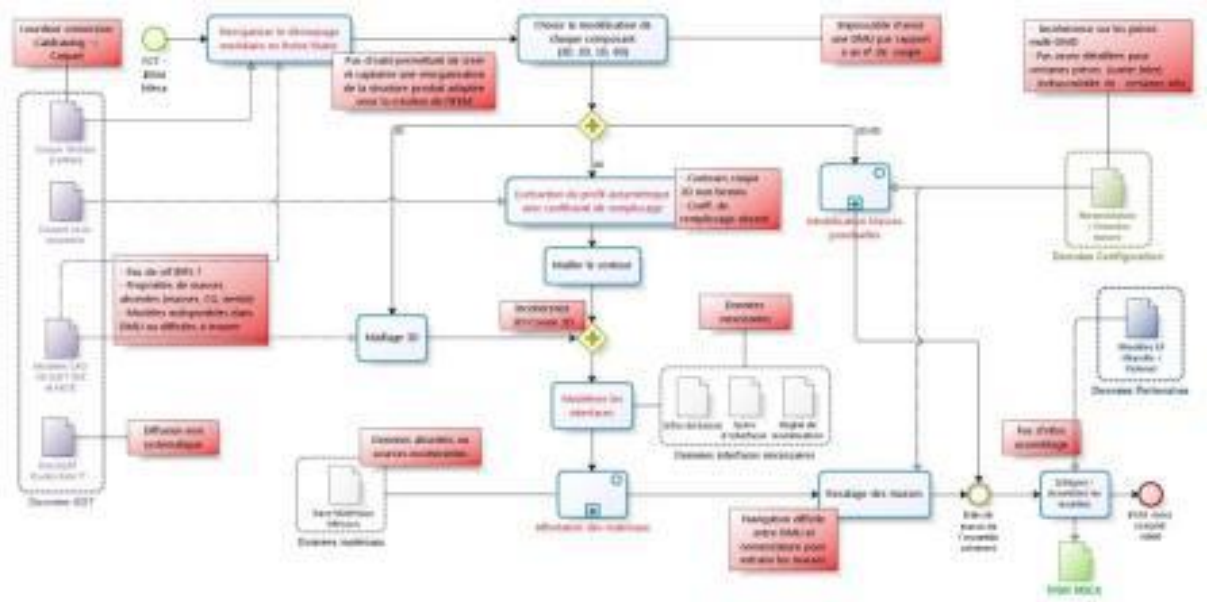


Figure 107: Example of VSM performed within Snecma Integration Division

9.4.2 Measure and Analyze

The Measure phase consists in defining the measurement system and the appropriate metrics and key performance indicators for the studied process. These metrics must be consistent with the type of performed activities, the type of value (process, product or information value) and the type of stakeholders considered. The definition of this measurement system and related metrics was based on the product development waste drivers mapping performed by [Bauch, 2004] and provided in appendix 3 (Figure 245). This measurements system and related metrics are not presented in this PhD thesis because their relevancy could not be assed and validated on operational processes. However it has enabled us to question ourselves when trying to define if an activity is a full value-added activity, a partial value-added activity, a necessary no-value-added activity or a non-necessary no-value-added activity. Moreover this measure phase has permitted to better prioritize our required capabilities in the Analysis phase and to better assess or identify the proposed solutions impact or benefits in the verify phase.

The Analysis phase consists in identifying the root causes of the waste or issues identified in the VSMs and to classify and prioritize the functional requirements that will be addressed. The results of the Analysis phase have lead to the industrial issues and requirements that have been introduced and summarized in Chapter 4.

9.5 Design and Verify phases: development methodology

Our development methodology is based on a classical software system engineering method structured as a V model and a UML approach for the software specification and design (see Figure 108).

The design top-down side of the V corresponds to all the specification and design phases. From the results of the VSM analyses of the studied processes we have identified the main waste and waste driver in digital integration chains. We have used the results of these analyses to express the high level objectives and functional requirements of the required design-analysis system integration framework. Performing a detailed functional analysis (using SADT methodology and formalism) we could define the required detailed functions, the software and environment architecture of framework as well as the information flow cartography. Then we have defined and developed our concepts and solutions proposal leading to the conceptual model. In the detail design phase we have enriched the data model with required attributes, links and methods and, when necessary, simplify it for easier implementation.

The implementation phase consisted in generating an empty data base from the logical data model, filling it with test-cases data, coding the specified object methods and designing and coding the GUI of the data base client application that will host our framework.

The integration phase (ascending branch of the V) consists in progressively testing the developed capabilities/functionalities individually and eventually re-iterating on data structures and/or object methods definition. When individually validated, a generic scenario of digital integration chain can be proceeded to validate the methods sequence and the usage of the framework within a business process. The cycle ends by the provision of a β-test solution and related documentation and their assessment by future potential users to validate the capabilities and/or make change recommendations.

The following chapters 10 and 11 summarise the results of the design phase; i.e. the functional synthesis, the description of concepts and related capabilities and the explanation of the conceptual data model.

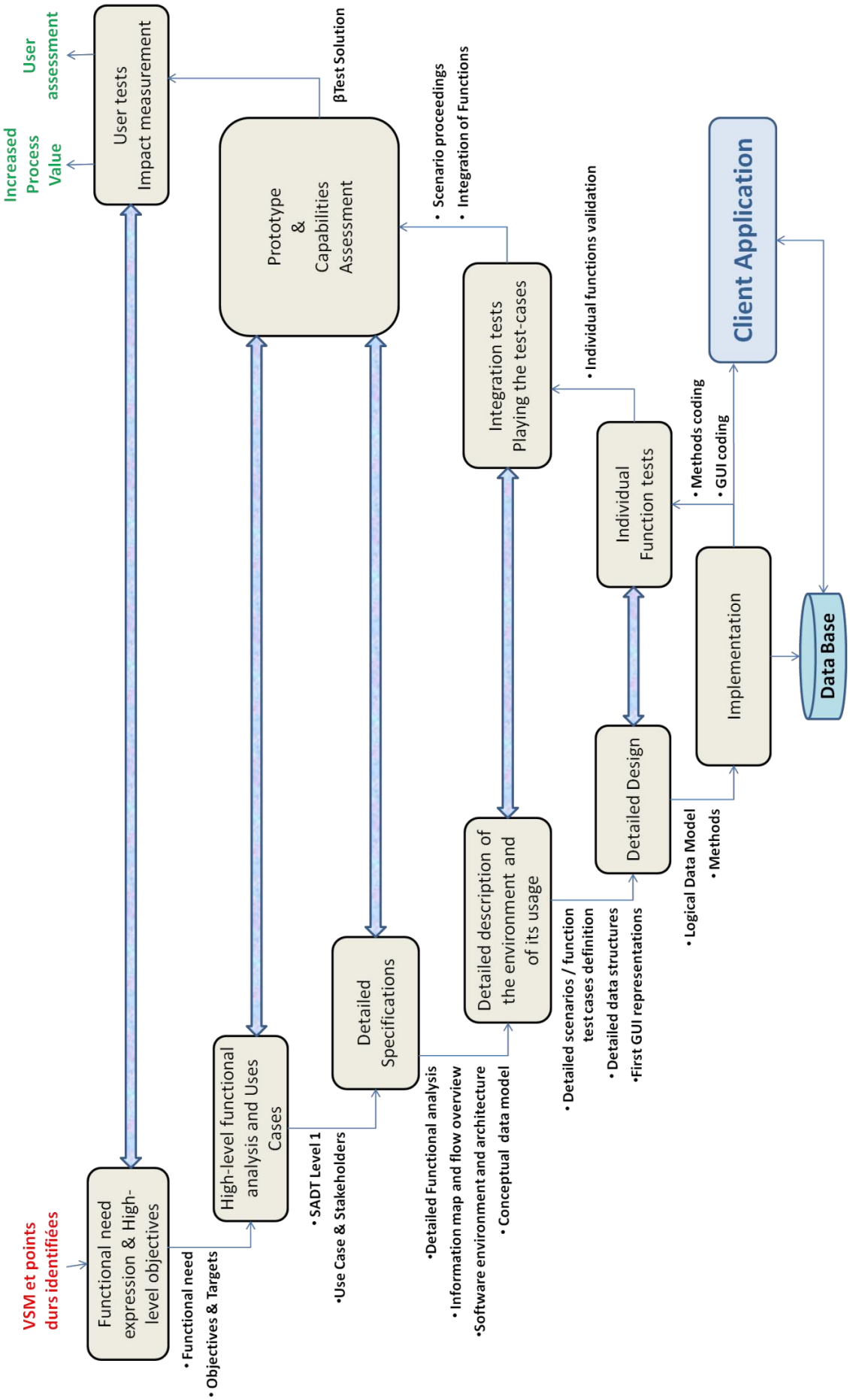


Figure 108: V-model development method

Chapter 10: Proposition of a MBSE framework for Design-Analysis Integration

This chapter addresses the specification of a multi-view and multi-disciplinary DMU and product data manager for simulation-based design. This framework is the major component of the federated and model-based environment required to answer the industrial requirements mentioned in Chapter 4 as well as fill the design-analysis integration issues and related gaps highlighted in the state of the art. We have called this environment the **D**esign-**A**nalysis **S**ystem **I**ntegration **F**ramework (**DASIF**). The objective is to define and develop a PDM prototype which functionalities and supporting data model enable to manage "multi-level" and "multi-domain" logical and physical (DMUs) product views enriched with assembly information in order to provide to analysts and integrators relevant data structures and inputs for creating FE assembly models.

This chapter is made of 6 sections. First section provides the results of the performed functional analysis including the general objectives of DASIF, the place of this framework in the product development process, the UML use cases of DASIF and the related list of involved actors and business roles, the conceptual and software environment architecture. Another sub-section specifies the structure and the representation of the various information and data which are conveyed through this environment and the functionalities of DASIF are synthesized. The question of its integration with other PDM and SDM systems but also with CAD and CAE environments is also tackled. Finally a conclusion provides a synthesis of this chapter. The second part of this chapter introduces the proposed approaches and related scenarios to exploit DMUs within DASIF. It also provides a list of conditions and required capabilities to support these scenarios. The third section is dedicated to our concepts and solutions proposal to provide some of these capabilities.

10.1 Results of the Functional Analysis

10.1.1 Objectives and fundamental need expression

The objective of the DASIF environment is to provide to system architects/integrators, designers and analysts a product data management systems and an "integration framework" enabling to identify, gather, acquire and organise the relevant data set used for simulations (particularly assembly FEA). The various digital product definition mediums and representations (bills of materials, digital models and system architectures) should be able to be derived and adapted according to the different views of the product used during the product development lifecycle and across the various engineering domains or disciplines. Hence, the resulting product data model should help to integrate various product views and gather the appropriate design data set relevant for specific needs and objectives of the simulations and related involved actors (depending on the life cycle stage, the discipline and the level of detail and abstraction required).

Figure 109 shows a bull chart (defined according to the functional analysis APTE method [De la Bretesche, 2000]) used to clarify the need of such an environment.

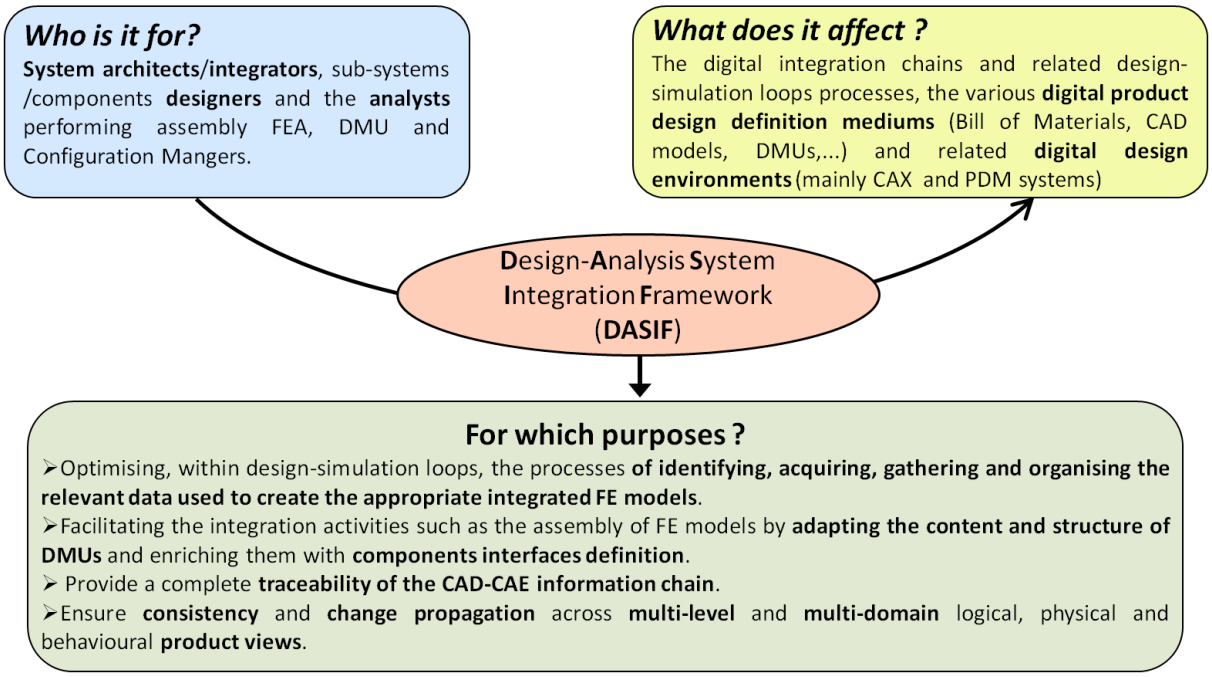


Figure 109: Fundamental need expression using the APTE method bull chart

The top-level function of DASIF is represented in SADT/IDEF0 formalism in Figure 110 below.

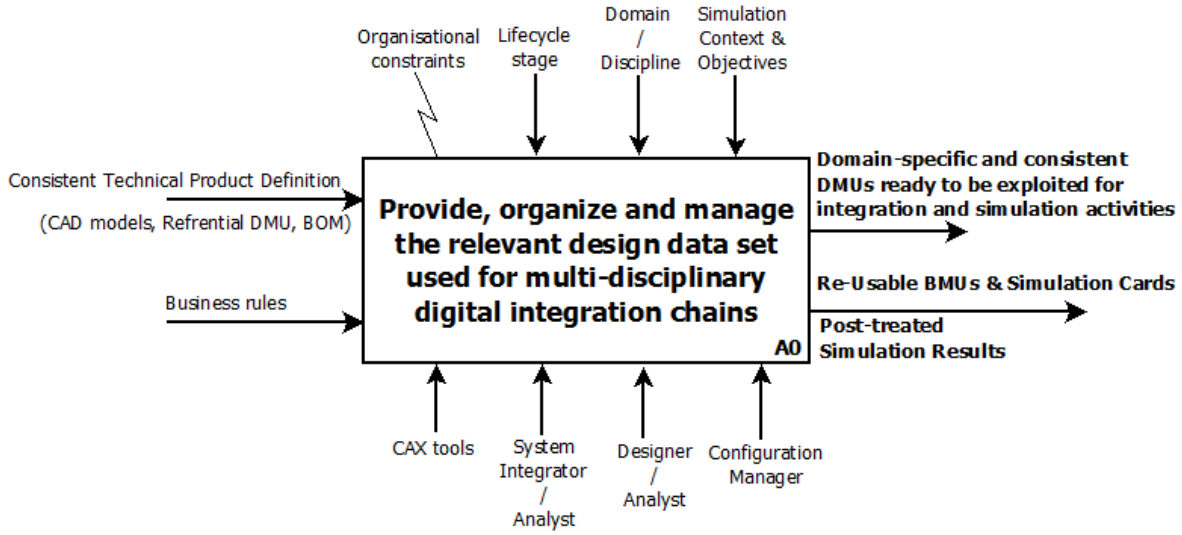


Figure 110: SADT/IDEF0 representation of DASIF top-level function

Structured Analysis and Design Technique (SADT) is a diagrammatic notation designed specifically to describe systems as a hierarchy of functions defined as activities of the system under specification. These functions are represented by activity building blocks and a variety of arrows to relate these functions/activities. In the boxes the name of the process or the action is specified. On the left-hand side of this box, incoming arrows represent the data inputs or consumables that are needed by the activity. On the upper part, the incoming arrows (or lightening for constraints) represent constraints, commands or data which influence the execution of the activity but which are not consumed. On the bottom of the box, incoming arrows (called mechanism) identify the means, components or tools used to accomplish the activity. Finally, on the right-hand side of the box, outgoing arrows identify the out-puts data or products that are produced by the activity [Marca&McGowan, 1993].

In order to describe the functional scope and packages of DASIF we have divided the top-level function A0 (see Figure 110) according to a classical Simulation-Based Design and Integration process as it generally occurs in the aeronautics industry. As illustrated on Figure 111, this process should follow the following steps:

- **A1 - Build & Manage System Architectures and related multi-view DMUs:** this functional package consists mainly in building the domain-specific product structures and DMUs that fit with the behavioural model structure. It includes the specification of components interfaces definition and their capture and visualisation within DMUs. This package should also ensure the consistency and synchronisation of the different product representations (product tree, system model, DMU, etc.) - corresponding to multi-domain and multi-level system characterisations (with several levels of abstraction) - with the product definition.
- **Provide the relevant design data set to use for simulations**: this functional package encompasses all the capabilities and methods developed to enable the analyst to retrieve the appropriate design data set to use for the simulation. This includes capabilities such as requesting and retrieving a specific DMU product view and access/identify/visualise components interfaces and interactions definitions, the capture and visualisation of CAD-CAE data links and the visualisation of design evolutions between two successive iterations.
- **A2 - Study Context Definition:** this functional package consists in defining all the information related to the study context: related product and project, base-line, objectives, type of study, involved disciplines, processes and methods to apply resources to use, etc. All these information will drive the way of performing the simulation(s) to perform for the study.
- **A3 - Sub-System FE Models Integration & Pre-processing:** this functional package uses the information defined in the two previous steps to specify, create, re-use, assess and exchange all the appropriate data that are needed to create and integrate efficiently the requested simulation models (potentially created by different partners) in order to deliver the integrated simulation model ready to be set-up for the computation.
- **A4 & A5 - Analysis Set-up and launch & Results data management:** once the computation done, this functional package aims at providing services for the verification of the obtained results, the analysis and post-treatment of these results and finally the dissemination of specific interfaces results to partners and/or co-designers for downstream design and simulation activities. This functional package must also ensure the traceability of the results with the study and related data/models that permitted to obtain them.

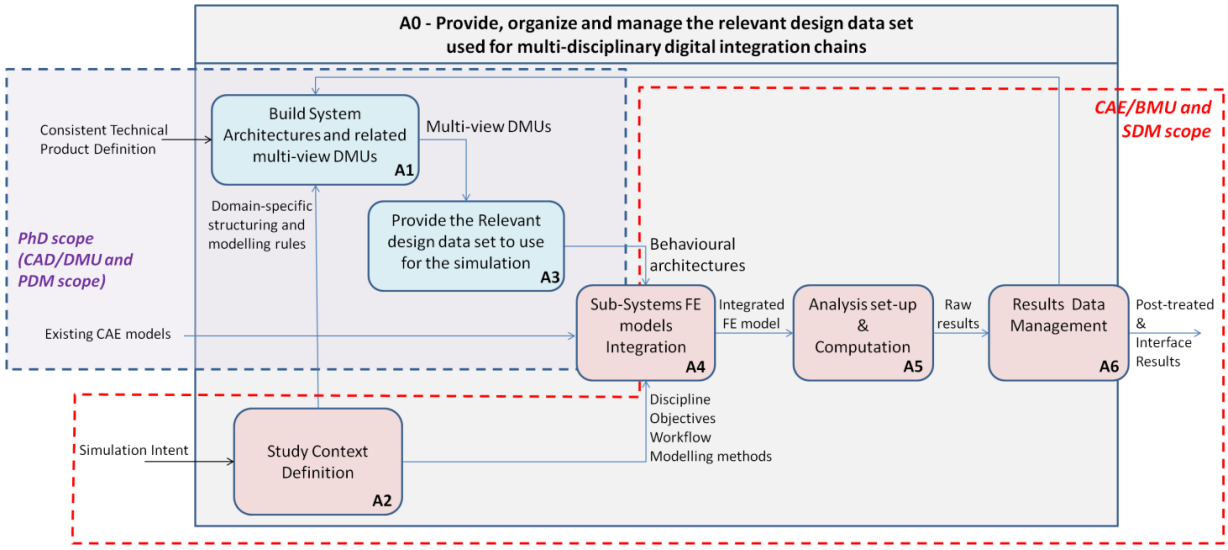


Figure 111: Simplified SADT of DASIF and positioning of PhD scope

10.1.2 Place of DASIF in the Product Development Process

The usage of the DASIF framework occurs all along the product development process to support digital integration chains and related design-simulation loops. As shown in Figure 112, it is hence situated between the CAD design modelling activities leading to the constitution of product DMUs and the CAE modelling and analysis activities leading to the creation of BMUs and related simulations results. DASIF also takes part of the configuration management process in order to configure these DMUs and related BMUs and maintain their consistency to the current product definition. Figure 112 shows the interactions of DASIF with CAD design modelling process activities, with Configuration Management Process activities and a classical FEA process. In this figure, the concept of **Multi-view DMUs** is introduced. They correspond to the domain-specific and simulation-oriented DMU product views that have been built through content and structural transformations of referential DMUs. They might be used as direct inputs for FE modelling activities and be consistently associated to their original DMU and the integrated FE model and simulation that they have permitted to perform.

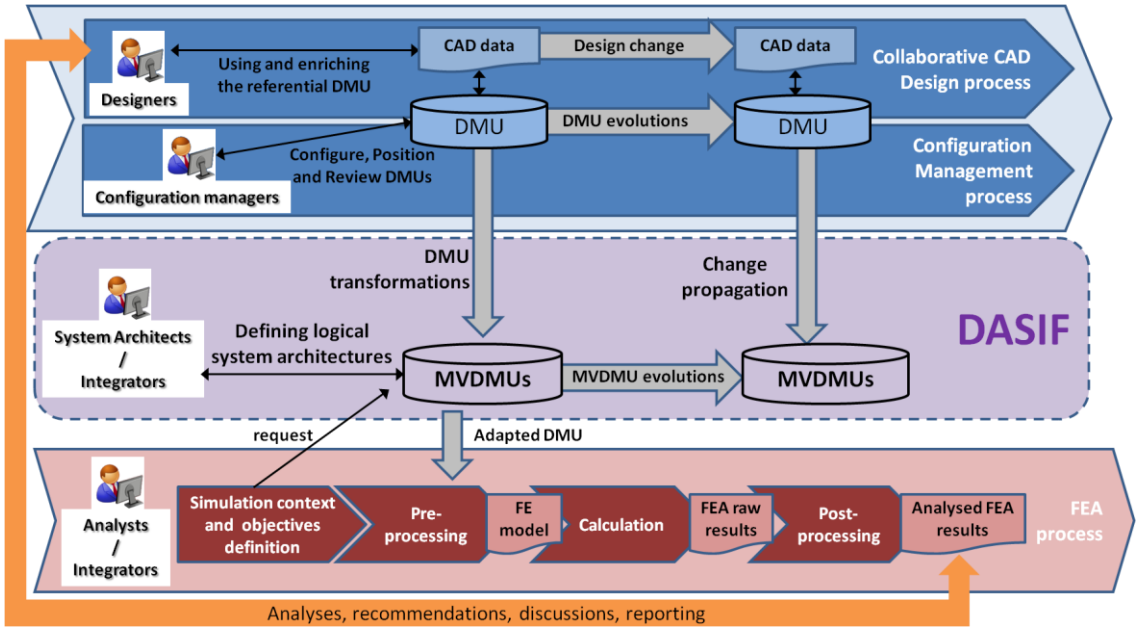


Figure 112: Position of the DASIF framework in the Product Development Processes

10.1.3 Stakeholders and Use cases

The UML use case diagram (Figure 113) summarises the main capabilities ensured by DASIF. In this diagram the involved stakeholders are represented. As well as a traditional PDM system, DASIF offers different functionalities depending of the respective role(s) of the various stakeholders or users of the application. A user can have a specific role or several roles depending of the context and the organisation he belongs to. For instance a designer can also perform FEA and the analyst or “simulation engineer” can be considered as a designer. An integrator can also be considered as a designer. We propose to speak about roles instead of speaking about different actors and we have defined a role taxonomy related to the various DASIF stakeholders:

- **The User**: it is the top-level role that represents any kind of user of DASIF with no specific attributions and rights.
- **The Administrator**: this "Super" user owns all the rights. He can create and delete users, define their rights and configure all system variables. He has access to all the data bases and can

manipulate, modify, create and delete any kind of data or meta-data managed by the application.

- **The Designer**: A designer can act as whether a simple design engineer, a system architect, a system integrator or as a simulation engineer or analyst. The designer role has all the rights apart from those of the Administrator to configure the DASIF environment and those from the Configuration Manager for managing the engineering change process. The Designers act upstream the simulation processes. They define and provide most of the technical product definition information/data that serve as input for simulations such as the 3D CAD models and related parameters and features, the components physical properties or the system architectures definitions. The designers can also access the simulation results and validate or invalidate their suitability to the functional requirements that the analyses intended to verify.
- **The System Architect**: this actor is in charge of the global system behaviours (e.g. thermal, mechanical, aerodynamic, etc) and of ensuring their compliance with related functional and performance requirements. The architect develops the behavioural architectures (with interfaces between components), matures these architectures by requesting new simulations. He makes decisions concerning global compromises between conflicting design parameters verified by different domain-specific simulations and concerning optimisation of behavioural architectures. He also commits on certification requirements fulfilment and their verification through physical tests. The architect is “mono-system” (reports to a given “Programme Chief” or “Chief System Engineer”) but he can be mono or multi-disciplines (e.g. in charge of thermal aspects or in charge of all physical aspects). The system architect has the same rights than a Designer and an Analyst plus specific dedicated “integrator rights” such as the definition and modification of system architectures.
- **The System Integrator**: this actor is similar to the system architect and has the same rights. However, we decided to attribute to this actor the responsibility for defining and monitoring the appropriate simulation and integration processes and related activities and actors. He designs the network of work tasks and allocates work tasks to the various involved co-designers and industrial sub-contractors. He is particularly in charge of defining and monitoring the related simulation workflow (design-simulation loops organisation, iteration duration, milestones and deadlines, resource allocation (load balancing), nature and format of data and design deliverables to provide). In the frame of DASIF and digital integration chains, the integrator specifies physical/functional (in relation with the architect)/behavioural interfaces between system components in an integration perspective. He also in charge of the validation of technical data provided for the integration such as the compliance of sub-system models with interface specifications or the simulation results distribution for downstream applications (e.g. feedback to design or downstream simulations).
- **The Configuration Manager:** the management of product configurations generally involved many functions and actors within a company. Here the configuration manager represents the actor responsible for defining the “master” product structures and related variants (from preliminary design to production and “in service” phase). He is also in charge of the management of product interfaces; maintaining the matrix interfaces in terms of coding, but also making interface data available to partners sharing the same interface. This actor needs to follow the technical definition of these interfaces since he is also responsible for monitoring the engineering change process and for ensuring that the validated design changes are properly propagated through the appropriate product configurations and related product views.
- **The Analyst or Simulation Engineer:** This actor retrieves through the DASIF application all the necessary and relevant design data sets required for the simulation activities of which he is in

charge. He uses these data sets to create or re-use the appropriate simulation models regarding the simulation context and objectives. Within DASIF, the analyst can request and load a specific DMU product view relevant for his simulation context and objectives. He can visualise and compare the data inputs used for two successive iterations to know which data need to be modified/updated. Within the requested DMU, he can access, identify and visualise the various component interfaces definition/specifications that allow him to make the right modelling assumptions and to assemble easier the sub-system FE models to integrate. The analyst also needs to visualise the complete CAD-CAE information chain to know, for instance, from which CAD model a certain FE model has been created. He can hence be notified if a change on a CAD model can potentially impact the FE model that he has created/used to perform a simulation.

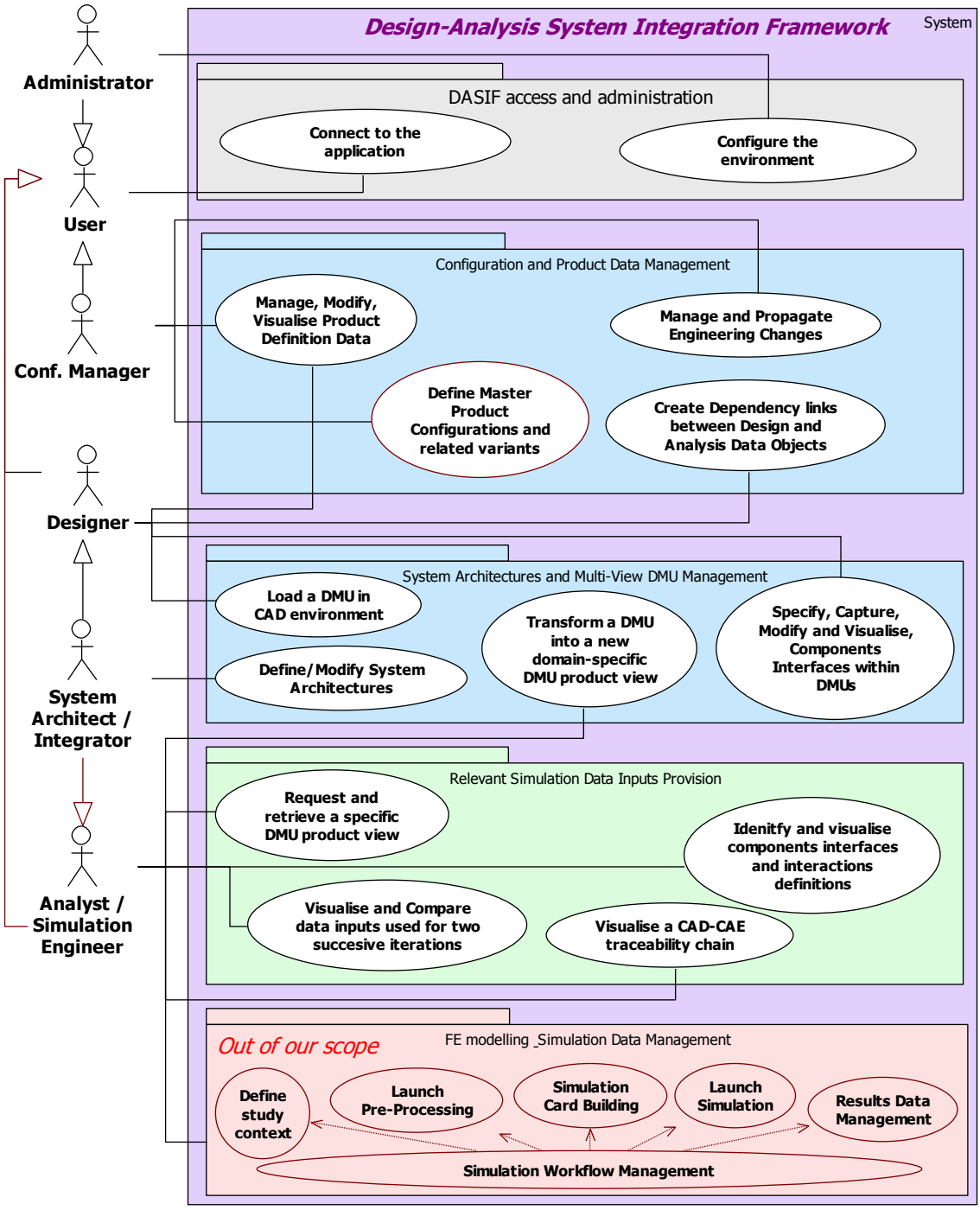


Figure 113: Simplified use case diagram of DASIS environment

In addition to this list it is important to underline the role of the **Programme Chief or Chief System Engineer:** he is the chief engineer in charge of programme targets (cost/delay/quality). He does not appear on the use case diagram because it is not planned that this actor uses the DASIF framework. However, this actor is generally involved in digital integration chains in order to give system technical orientations but also to arbitrate between System Architects conflicting alternatives.

NB: In the use case diagram shown in Figure 113, a functional package called “Simulation Data Management” (SDM) is characterized by an annotation “Out of our scope”. This SDM package is subject of a different, yet complementary research study related to Graignic’s PhD as well as in collaboration with CRESCENDO project partners [Graignic et al., 2013]. Through the SDM environment, the analyst has access to all analysis data and meta-data; i.e. simulation workflows, simulation context and objectives, FE modelling assumptions and specifications, mesh models, load cases, boundary conditions, and simulation results. As it is within the SDM environment that the simulation workflows are defined and monitored, simulation activities inputs and outputs can be associated in order to access and visualize the complete CAD-CAE traceability chain (links between nominal, idealized CAD models, design parameters, simulation models and simulation results). That is why, even though it is out of our research scope, the SDM package and related data models and structures need to be considered when developing DASIF.

10.1.4 Conceptual and software environment architecture of DASIF

DASIF is a federated digital design framework dedicated to the needs of digital integration chains activities. It provides a combination of PDM/PLM and SDM/SLM capabilities as well as a federated MBSE system modelling framework for managing system architectures and enhance design-analysis integration.

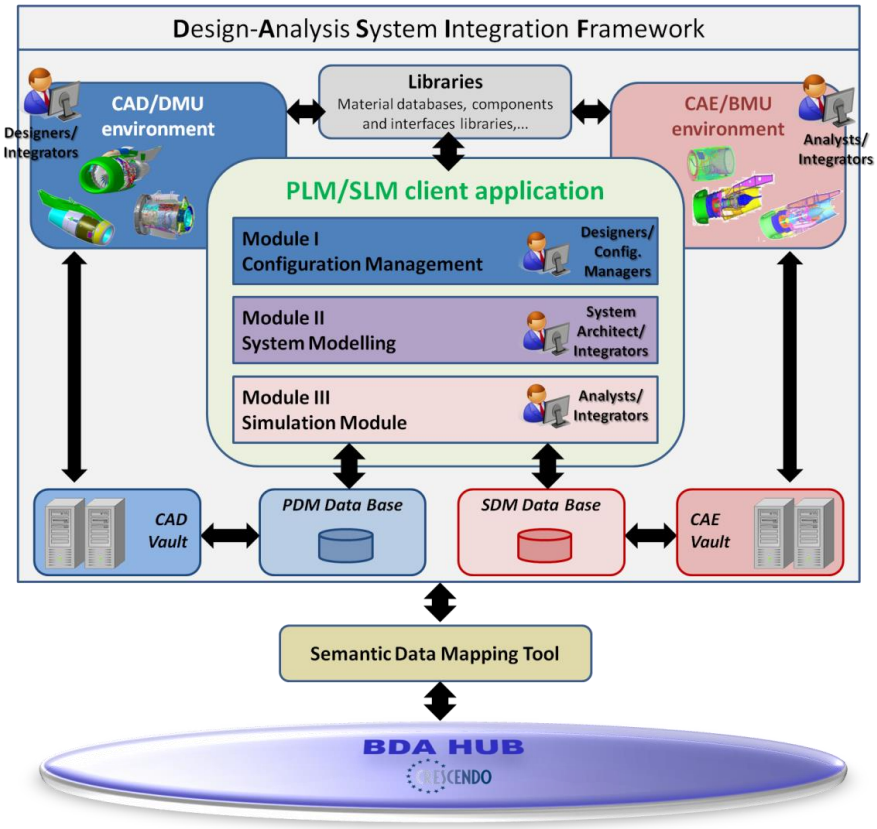


Figure 114: Conceptual scope and architecture of DASIF and link with the BDA

As shown in Figure 114, DASIF encompasses the following digital design environments:

- **PLM/SLM client application**: in the frame of DASIF, this application encompass three main functional modules which are:
  - **The Configuration Management Module**: managing the creation, modification, versioning and access of all design product data and their configuration according to multi-level and multi-domain product definition views. This module is also in charge of maintaining the consistency between these product views and related representations as well as ensuring engineering changes propagation all along the development lifecycle.
  - **The System (Modelling) Module**: managing the system models enabling definition and specification of structural and behavioural architectures of the system using an object-oriented system modelling formalism. The dependency links between product views can also be defined in this module.
  - **The Simulation Module**: encompasses all capabilities dedicated to the management of simulation workflows defining study contexts, building the simulation cards and launching the computations (i.e. putting together the meshes and boundary conditions submitting it to a specific solver), getting and storing the analyses outputs and preparing the analysis report.
- **CAD/DMU environment**: relates to all CAD software applications permitting to create, modify and visualise the CAD product models.
- **CAE/BMU environment**: relates to all CAE applications (pre/post-processing tools and solvers) enabling to create, modify, visualise, simulate and analyse CAE models and results.

The PLM/SLM client application interacts with both a PDM data base and a SDM data base enabling the storage and re-use of CAD and CAE models/results in, respectively, the CAD and CAE vaults. The PDM data base is built upon a product data model defined in Chapter 11. The SDM data base and related data model are out of the scope of this PhD and might be defined within another research work.

Moreover, the concepts and capabilities proposed for the DASIF framework can also be implemented in the BDA digital collaborative platform in order to integrate and synchronise design-simulation processes between partners and to exchange/share design and analysis data/models throughout the development life cycle of aeronautical products. The BDA platform is built on a standard data model enabling interoperability, to which existing local design environments and new services to be developed could plug. To enable this interoperability the use of a semantic data mapping tool is proposed. This tool aims at mapping the proprietary PLM/SLM data models and formats of commercial applications to the BDA standardized business object model.

Figure 115 below proposes another representation of DASIF architecture underlining the central role of the “System Module”.

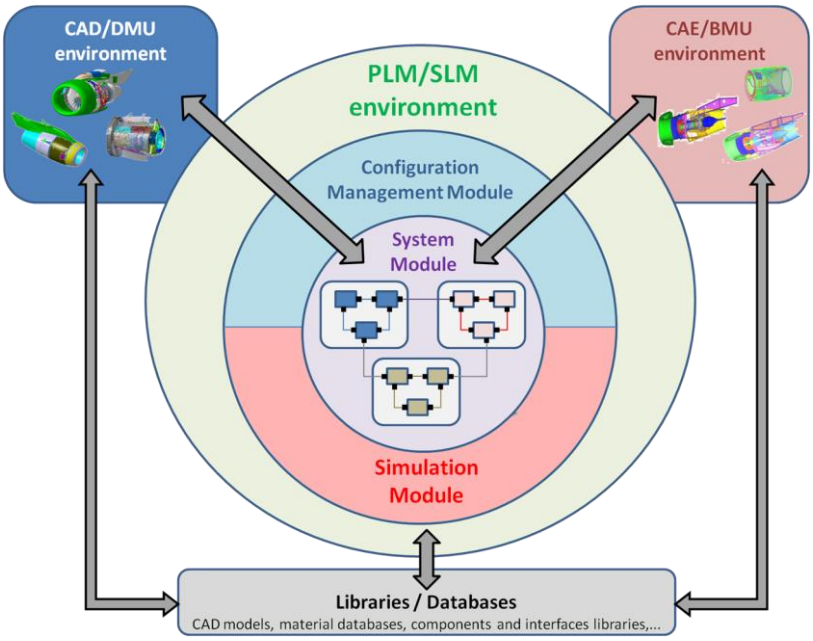


Figure 115: Other representation of DASIF architecture - the System Module cornerstone

This system module is a MBSE framework that uses an object-oriented system modelling language (similar to SysML) as the front-end for multi-disciplinary teams to collaboratively develop a unified, consistent representation of the system from the earliest stages of development. The system model (in SysML) can 'co-evolve' with the associated domain-specific models, such as CAD and CAE models. Relationships between the system model and the domain-specific models can range from qualitative dependency relations to quantitative parametric relations which are executable on-demand for seamless model traceability and interoperability. The SysML-based system model is a conceptual abstraction of the system that has sufficient details for orchestrating digital integration activities, ranging from structural and behavioural architectures definitions (including domain-specific interfaces specifications), automated structural DMU transformations and FE assembly operations, access to appropriate design and analysis models through the use of building blocks and components and interfaces model libraries. This unified system model is not a data store but a description of the system which can be used to federate domain-specific models of different aspects (links between interdependent product views used at different system breakdown levels or for different design disciplines) of the system.

10.1.5 Information flow map

The Figure 116 shows all activities and related information/data flows taken into account by DASIF. It enables to identify and visualise all activities inputs and outputs, hence all the business object and related data objects that are conveyed between the DASIF framework, EDM systems (PDM and SDM data base), the CAD/DMU environment and the CAE/BMU environment. The activities that take place within the PLM/SLM client application of DASIF are the ones contained in the central green area. The other activities can take place in the CAD/DMU environment or in the CAE/BMU environment.

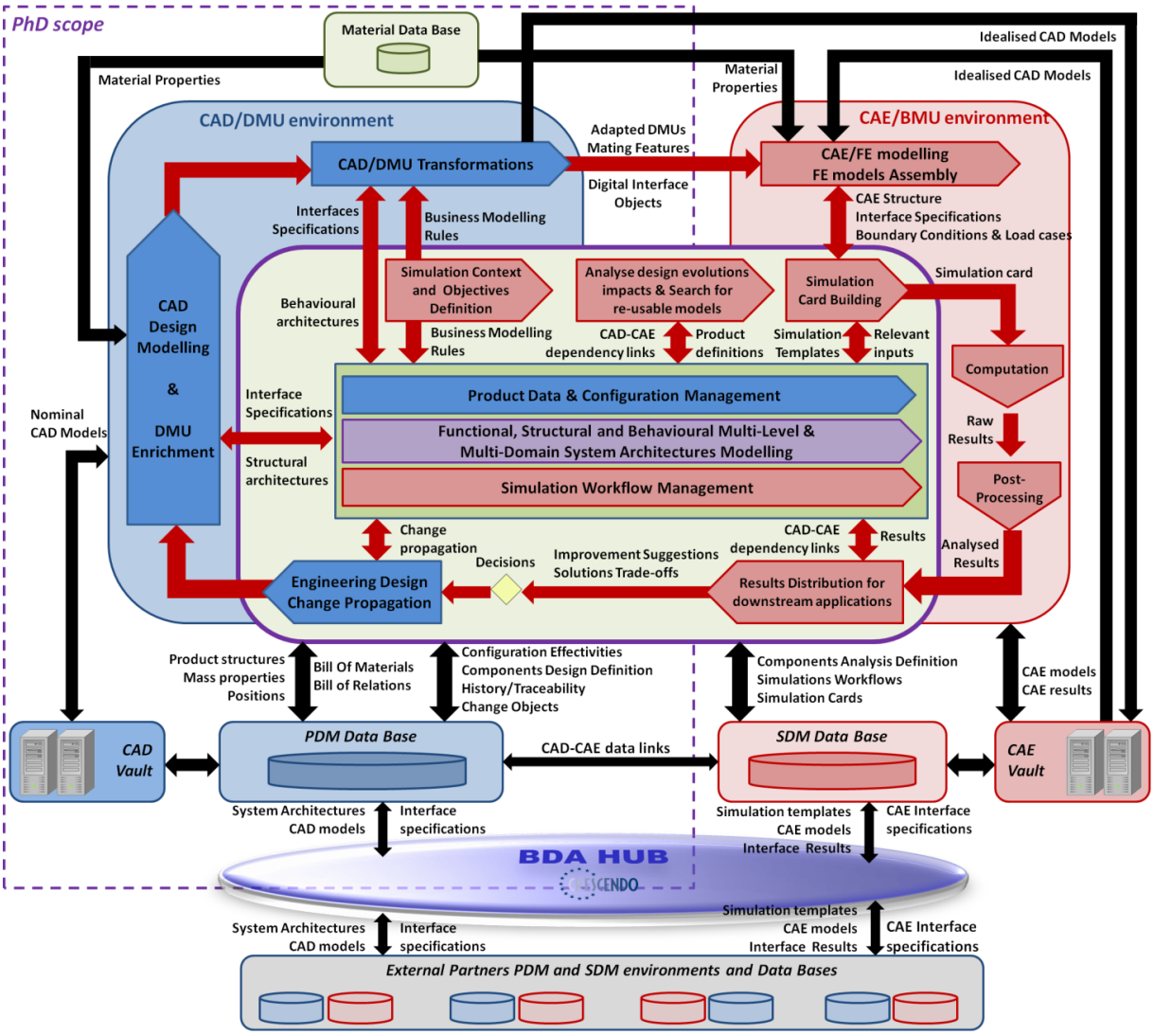


Figure 116: Map of Design-Analysis Information flow within DASIF and within a design integration chain

This information flow chart is built around the key digital integration chain activities which are:

- **CAD modelling and DMU enrichment activities**: DASIF will impact these activities by enabling to configure DMU content and structures. It will also provide a support for creating and managing interface digital objects within DMUs in order to support design in context, the automation of FE assembly modelling activities and potential automated DMU structure re-arrangement. These activities take place in the CAD/DMU environment but use and/or enrich the information present in the DASIF client application; especially the ones managed by the Configuration Management module and the System Modelling module.
- **System architectures definitions**: these activities might take place in the "System Modelling" module of the PLM/SLM client application of DASIF. Within this module DASIF must offer an integrator dedicated environment that permits defining and relating functional, structural and behavioural architecture definitions with the use of an object-oriented system modelling formalism. The SysML-based system model is a conceptual abstraction of the system that has sufficient details for orchestrating digital integration activities, ranging from structural and behavioural architectures definitions (including domain-specific interfaces specifications), automated structural DMU transformations and FE assembly operations, access to appropriate design and analysis models through the use of building blocks and components and interfaces model libraries. This unified system model is not a data store but a description of the system

which can be used to federate domain-specific models (links between interdependent product views used at different system breakdown levels or for different design disciplines) of the system.

- **CAD/DMU transformations**: These activities take place in the DMU environment but use the information displayed by the DASIF client application. When running a simulation, not all DMU parts are required: there is a geometric zone of interest. Moreover, the implicit "groups" created by the tree chosen to structure the DMU is often not the best choice for structuring the simulation inputs. As a result, there is a need of additional layer on top of DMU, which allows reorganizing the DMU for simulation purposes. Therefore DASIF might offer capabilities to filter a DMU and keep the parts present in the geometric zone of interest, automate or semi-automate the reorganisation of DMU structures and to manage consistently the resulted discipline-oriented DMU configurations.
- **FE assembly modelling**: These activities take place in the CAD/DMU environment but use the information present in the DASIF client application. These activities bridge the DMU geometry and the mesh used inside the simulation. Within the DASIF client application the simulation card - gathering components and interfaces FE models, boundary conditions and load cases – is built based on simulation templates and on the structure of the assembly FE model corresponding to a behavioural architecture defined in the system modelling module. Within the CAE/BMU modelling environment, DASIF must provide automatic routines to translate the system behavioural model into an authoring tool format/language and import it in the dedicated pre-processor to automate the assembly of FE models. In this layer, DASIF might also handle the complexity of the geometries contained in the DMU, and serve the users of meshes of fit-for-purpose size and quality, involving idealization procedures. DASIF should integrate in new discipline-oriented DMU configurations the simplified and idealised CAD models used to create assembly FE models and relate them to their original nominal CAD models.
- **Simulation related activities** (out of scope but considered): this concerns all simulation activities downstream the pre-process. It encompasses submitting the simulation card to a specific solver, getting and storing the analyses outputs, and preparing the analysis report. Several analyses may be chained, and/or the same analysis may be done several times, with different parameter values. This is driven by the module "simulation workflow management" which is a pre-requisite for putting robust design/optimisation on top. The simulation results distribution for downstream activities can be supported by system models associating interface results to corresponding interface objects (ports) in the system model.
- **Engineering Change Propagation:** After an engineering design change is decided and performed, all information related to this change is gathered within a configured change object. Within DASIF this change object and its configuration serve to propagate this change in the appropriate product configurations and related representations (product trees, DMUs, system models).
- **Standardized data exchange through the BDA:** since the product is developed in a collaborative and distributed design environment where sub-systems are designed at each partner site, DASIF should be able to communicate with other different PLM/SLM systems via a digital collaborative platform. The BDA hub represents this collaborative platform. It is supported by a standard data model and an IT infrastructure to which existing local design environments and new services to be developed could plug. In the functional scope of DASIF the data/information exchanged through the BDA are: system architecture definitions including interface definition, CAD and CAE models associated to the building blocks of the architectures as well as related modelling specifications.

10.1.6 Functional synthesis

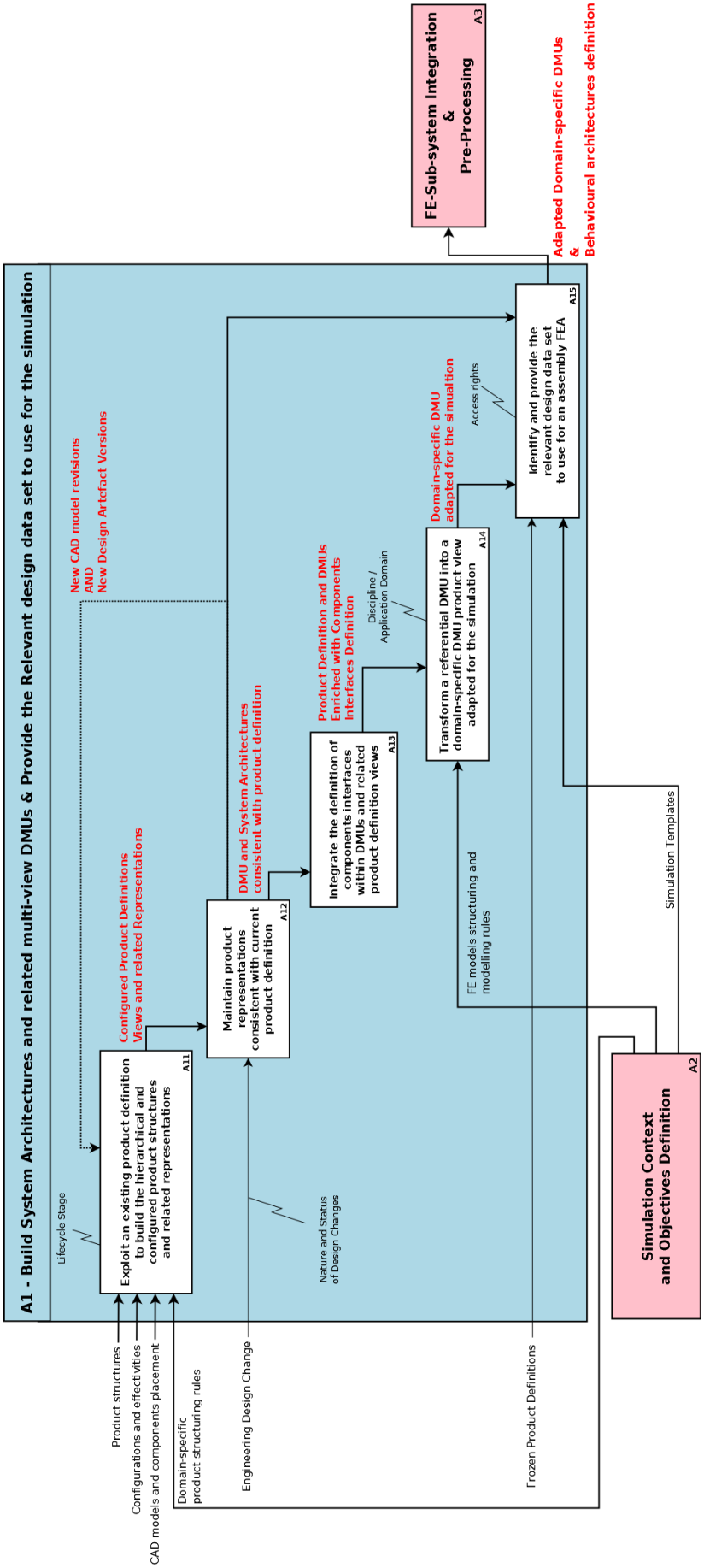


Figure 117: DASIF A1 SADT decomposition – High level functions to provide

The objective of our work was to study the way to define and organize the relevant design data sets used for simulations regarding the objectives and the type of the analysis, in order to reduce the time for the creation/integration of simulation models and also to ensure exchange between partners in a collaborative environment. As shown on the above SADT/IDEF0 diagram of Figure 117, the functional analysis of DASIF lead us to gather A1 and A3 in the same functional building block and to breakdown the top-level function A1 -"Build System Architectures and related multi-view DMUs & Provide the Relevant design data set to use for the simulation"- into the five following principal functions that are detailed in this section:

- A11: Exploit an existing product definition to build and provide the hierarchical and configured product structures and related DMU and system representations;
- A12: Maintain product representations (DMUs, system models, BOMs, 2D drawings) consistent with current product definition;
- A13: Integrate the definition of components interfaces within DMUs and related product definition views;
- A14: Transform a referential DMU into a domain-specific DMU product view adapted for the simulation;
- A15: Identify and provide the relevant design data set to use for an integrated assembly FEA.

The A3 function – "FE sub-systems Integration & Pre-Processing" – is also partially detailed because it encompasses two important steps and functions regarding the PhD scope: the automated assembly of FE models and the exchange of system architectures and modelling specifications through a federated digital collaborative platform.

10.1.6.1 A11: Exploit an existing product definition to build and provide the hierarchical and configured product structures and related representations

Identified need: currently product data are neither managed consistently regarding the design context for which they are relevant, neither controlled through an integrated reference framework that permit to display the appropriate view of the product to use in the specified context. Different views of a system all relate to the same system and thus depend on each other. In addition to integration between domain-specific views, a mechanism for multi-level system design is required to ensure the continuity of information between views at different system breakdown levels. This integration is difficult to handle due to a lack of systems-level modelling capabilities. The integrators might need to specify and design these product views and their links through the use of an object-oriented language enabling to specify system blocks interactions and interfaces but also the links between product views. The designers and integrators might be able to visualise and exploit a DMU and corresponding System Model representing the appropriate product view to use in the specified context.

Functions & capabilities to develop:

- A111: Import an existing product definition
- A112: Configure product definition views applying effectivity rules adapted to the lifecycle stage of the product and the application domain of the product view.
- A113: Represent and organise a product configuration (assembly structure) into a hierarchical tree structure and build the equivalent system representations.
- A114: Create/Specify and Modify system architectures with an object-oriented formalism.
- A115: Create dependency links between system architectures and related elements
- A116: Apply identified effectivities and obtain the exact content of product configuration

- A117: Create, Modify or Delete effectivities of product structure links
- A118: Relate appropriate CAD models to corresponding parts and ensure their relative positioning according to the reference frame of the parent component in the configuration
- A119: Load a configured and positioned Digital Mock-Up in a CAD modeller application

10.1.6.2 A12: Maintain product representations consistent with current product definition

Identified need: In order to be considered and used as the product definition referential, it is essential for a DMU environment:

- to be consistent with the current product definition and to integrate progressively and consistently the engineering design changes occurring during the development lifecycle;
- to enable representating a system from multiple viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail
- to be able to generate from the DMU assembly definition the other corresponding product representations (BOMs, system model, 2D drawings)

Current EDM systems require intelligent objects and operations enabling to provide the suitable and reliable CAD and DMU data inputs regarding the specification and the purpose of simulation (mismatch between study requirements and data used to perform the simulation). They also must support interface definition and specification that include multiple levels of hierarchy and multiple level of abstraction in order to optimize the integration/assembly activities whether it is for contextual CAD design or for the creation of Integrated FEM (IFEM).

Companies require product data views for each department that support their specific views and engineering change (EC) propagation procedures that maintain consistent product data between design departments. For consistent EC management between multi-domain or multi-disciplinary design teams, ECs should provide product structure-oriented representations, and “effectivity” management for domain-specific product views, integrated objects for workflow applications and integration with product configurations.

Functions & capabilities to develop:

This function encompasses the following capabilities:

- A121: Identify, save and visualise the successive states of product or system components definition.
- A122: Save and Freeze the successive states of a product configuration and associate it to a simulation iteration/loop.
- A123: Be notified when an engineering design change impacts a product configuration for which the designer is involved
- A124: Visualise the nature and status of the design change
- A125: Integrate the changes to the current product definition and propagate them to the related product variants (domain-specific configurations) and representations.
- A126: Generate BOMs from DMUs
- A127: Generate 2D drawings from DMUs

10.1.6.3 A13: Integrate the definition of components interfaces within DMUs and related product definition views

Identified need:

In order to prepare the FE model of sub-systems interfaces and interactions, CAD systems are reduced to an interactive and intensive use of components surfaces cutting operations to define the contact areas and connect the resulted trimmed geometric areas. These manual operations represent a really time-consuming and tedious effort for analysts. Therefore, the preparation of FE models requires that the CAD product assemblies, represented in the DMU environment, contain the following information:

- The explicit representation of the geometry of the contact (or clearance) surface between the connections and the assembled components;
- In some cases (mechanical analysis), the type of kinematic linkage involved in these connections;
- The linkages semantic and technologic information (bolted flange, bearing, groove, rivets, suspensions, etc.) as well as the related specifications regarding the contacts or clearances required to ensure this technological linkage. The semantics is highly dependent on the system studied, on the semantic codes of the industrial sector and organisations involved or even on the language used by the organisations. The topology specification of a technological linkage might be captured by templates in order to re-use and re-instantiate them.

There is a specific need to integrate fluid domains and their geometries within DMUs in order to specify the interactions and interfaces between a fluid and a solid component or even between two fluid domains.

Functions & capabilities to develop:

- A131: Enrich PDM data base and DMUs with “Fluid Domain” components and related geometries
  - Create, integrate and manage “Fluid Domain” components within the relevant product configurations (e.g. thermal DMU)
  - Generate, save and relates to the appropriate artefact definition the fluid domain geometries: fluid/solid exchange surfaces, fluid domain envelope.
- A132: Create and integrate consistently components interfaces definition within product structures and within DMUs.
  - Create, define and save the various types of interface artefacts: managed like other system components but with specific definitions and attributes.
  - Integrate them consistently within product structures (to the right product configuration(s) and at the right breakdown level) and within system representations.
- A133: Identify and localise the area of interactions (CAD mating features) between system components within DMUs through the use of publications inherent to the CAD models and reference them within the definition of the corresponding interface artefact definition in the PDM data base.
- A134: Associate a technology and related CAD model(s) to the definition of an “interface artefact” when the interaction that takes place at the interface is ensured by a physical component.
- A135: Create and manage “standards interfaces” to store in a interface library/catalogue and to be instantiated while creating and specifying a new interface artefact

- A136: Exploit functional, physical and technological definition of the various components interfaces within DMUs to design in context the integrated product:
  - Access and visualise interface properties and related mating features
  - Automating the update of components placement when an interface is re-sized or moved.
  - For the “standards interfaces”, access and visualise the required interface template
  - Use interface templates to place an “interface component” CAD model representing the technology used to ensure the interaction

10.1.6.4 A14: Transform a referential DMU into a domain-specific DMU product view adapted for the simulation;

Identified need: Analysts often need that the organisation of the components represented in the DMU matches with the structure of their simulation model. It is obviously not the case in As-Is DMUs. Re-arrangement of DMU product structures is also a tedious, time-consuming but essential activity to prepare the simulation model. There is a crucial need to enhance automated or semi-automated generation of simulation-driven DMU sequences/structures/scenes.

Functions & capabilities to develop:

- A141: Generate a system or graph representation of an assembly from a DMU containing interface definitions
- A142: Reorganize components according to the functional and kinematic definition of their interfaces.

10.1.6.5 A2: Study context Definition

Identified need: the first step in simulation activities is to define the context of the study. Simulation has a purpose, which copes with the study of a behaviour of the product in order to validate, optimize, evaluate its performances. The type of study (behaviour and goal of simulation) implies specific information that will drive the way of performing simulation (type of model, type of simulation and solvers used, expected result). Regarding all the data implies in a simulation, they have to be organized and structured to know for what study these data has been used or created. To define a study context the following required information must be captured and retrieve:

- The purpose/goal of the simulation study: In the product development process, simulation is linked to a specific objective. This objective is to study the behaviour of the product in a defined situation, environment, in order to validate the design studies, or to specify the design of the product.
- The simulation type: Following the different goal of the behavioural study, a kind of simulation is used. Simulation could be an acoustic, thermal, mechanical, CFD simulation or an optimization, a multi-physic simulation, a distributed simulation. The type of models and/or the type of solver to use will be derived from the different defined simulation types.
- Product description: The simulation is performed on a specific baseline of the product, which are characterized by its configuration and version status regarding the development process. This information enables to select appropriate models linked to the configuration and version of the product. This product description is complete by the operational state of the product. This information provides each case study about the condition of use of the product.

Functions & capabilities to develop:

- Define a study / analysis context
- Define and re-use study/simulation templates
- Use the study context information to identify and provide the relevant data set to provide for the simulation

10.1.6.6 A3: Provide the relevant design data set to use for an integrated assembly FEA.

Identified need: product data models must support the consistent integration of design information from the product definition (and especially the CAD data present in DMUs) with the associated simulation data (FE models, results) in order to ensure a complete traceability of the design/simulation information chain and to enable the management of change impact in PDP. The analyst might be able to retrieve all the design data inputs used to create a simulation model. He also might be able to freeze a product definition and related views and to compare two similar product configurations taken at two different time t in order to analyse design evolutions. He might be informed about the nature and details of the design changes.

Functions & capabilities to develop:

- A31: Retrieve all the design data inputs used to create a simulation model and be able to trace all the CAD-CAE in information chain
- A32: Visualise and compare two similar product configurations taken at two different time t
- A33: Request and retrieve a specific DMU product view and access/identify/visualise components interfaces and interactions definitions

10.1.6.7 A4: FE sub-systems Integration & Pre-Processing

Based on a common shared product definition (a behavioural system architecture), and based on the study context definition, the goal of this functional package is to create an Integrated Finite Element Model (or assembly FE model). The main hypothesis is that we are working in a collaborative and distributed design environment where sub-systems are designed at each partner site. In that context, the integrated simulation model is built from the assembly of sub-systems simulation models specified by the integrator but created at partners' sites. Figure 118 below, describes the sub-functional packages that compose the "System Integration" package:

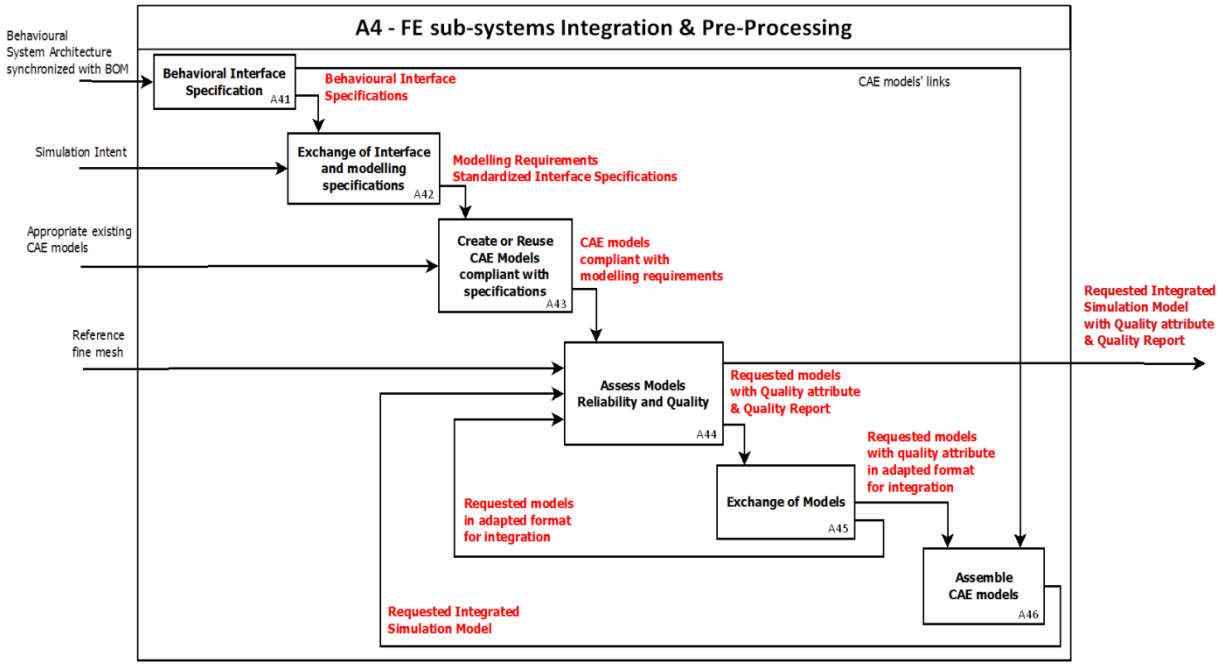


Figure 118: A4 SADT - SADT - FE Sub-systems Integration & Pre-Processing

Identified need

- Exchange and use of interface and modelling specifications for the integrator to be supplied with appropriate FE models of fit-for-purpose type, size and quality;
- Interoperate with CAE pre-processing applications to create or re-use the required FE models;
- Assess the quality of FE models to integrate;
- Speed-up the FE process by automating the assembly of sub-systems models;

Functions & capabilities to develop:

- A41: Behavioural interface and modelling specifications;
- A42: Exchange of interface and modelling specifications between heterogeneous PLM/SLM systems;
- A47: Automate the assembly of sub-systems FE models within the appropriate CAE pre-processing applications.

10.2 Proposed approach for exploiting DMUs in DASIF

10.2.1 Definitions

In the following sections, the reader might be confused by the use of complementary and close notions. It is hence important to provide a precise definition for each of these notions.

Referential DMU (RDMU): as defined in 3.3.2, a RDMU is a snapshot at a given time, i.e. at a maturity level, of a product configuration definition permitting to display its 3D representation in a CAD modeller or viewer. As its name suggest a RDMU is, for a certain time period, a product definition reference for deriving other related product representations (e.g. BOM, 2D cross-sections) and adapted Downstream DMUs. Therefore, the CAD models contained in a RDMU evolve during detailed design phase to an “as-manufactured” definition; i.e. with all constituting parts and with a high level of geometric details.

Downstream DMU (DDMU): A DDMU, and based on the definition from [Drieux, 2006], is a 3D-based product view gathering the just needed information for a given usage, i.e. a targeted downstream application (e.g. various multi-disciplinary finite element analyses, manufacturing sequence simulations, maintenance and accessibility checking, etc.). The DDMU is partially generated from data retrieved and/or adapted from a RDMU. However, a RDMU, even if more detailed, may not contain all the information and data required for the targeted application. Therefore, in addition to mechanisms of adaptation, the process of RDMU transformation into DDMU might integrate enrichment mechanisms.

DMU product view: A DMU product view is a restriction (or subset) and a specific data organization of another initial or referential DMU, defining together an understandable and appropriate representation according to a specific business perspective (e.g. the design view, as designed) or to a specific application. Therefore, all the generated DDMUs defined for a business or a given targeted application, allows defining new views of the product for this business or this type of application: a DDMU is a specific view of a RDMU.

Product configuration: Rule describing the composition and structure of a hierarchical system/product. Therefore configuration rules enable the definition of the structure and the relevant content of the various DMU product views.

Master product configuration: it is a product configuration that is established each time it is necessary to agree on a reference which is the basis for identifying further configurations and related product views.

Targeted downstream application: a DDMU is hence specific to a targeted downstream application. While a business may be linked to a specific product representation corresponding to this business, a downstream application defines a specific usage of this representation to access certain aspects of the product that may or may not be present in the RDMU. This application is defined by the user needs (in terms of structure, content and details of the data to be present in the product representation) regarding the objectives of this application but also by the constraints for generating and exploiting the related product representation. For a given business, a targeted downstream application is defined by:

- One **Discipline or Business:** defines the business or discipline (mechanical integration, thermal integration, acoustics, marketing, manufacturing assembly) in which the product representation is used;
- One **Use Case/Scenario:** within a same discipline, several different use cases of the product representation can exist (e.g. simulate the impacts of a physical phenomena, design in context, specify components interfaces, exchange of CAD data, etc.).
- One **Context** in which the use case is performed: the context refers to the product lifecycle stage, the system breakdown level, the considered product configuration and the role(s) of the user(s).
- One or several **Objectives**: defining the final targets of such a usage of the product representation (e.g. verify system compliance to design requirements X, Y, and Z).

DMU Scene: as defined in 7.2.2, the word “scene” is generally used in the context of visualization of 3D CAD models to identify all the elements/entities (3D objects, materials, attributes, etc.) needed

to visualize and to possibly handle. In a more generic approach, the scene can be considered as the set of data needed to achieve an application scenario, whether it concerns visual data, simulation models or various associated attributes. A scene only contains the relevant and significant data/information (entities and relationships) for this application scenario and also strongly refers to the organization of these data; which is itself linked to the targeted application scenario. The “scene” notion differs from the DDMU notion since it designates a representation of a RDMU or a DDMU which is directly accessible and exploitable by the end user, whether the scene is visual or interactive. The generation of a DMU scene correspond to the stage of translating the DMU data in the specific language and format of the modelling, viewer or simulation tools that make these data understandable and exploitable by the end user.

10.2.2 Proposed approach: from “Referential DMUs” to “Downstream DMUs”

Due to the complexity and variety of possible targeted applications and related use case/scenarios, it is impossible to provide a generic and valid solution for all these applications and use cases. Since our research work focus on the exploitation of DMUs for assembly FEA, the targeted downstream applications mentioned above will only concern different types of FEAs. In this context, the related use cases will correspond to the nature and type of the analysis (e.g. for the “mechanics” discipline: performing a mechanical static linear constraints analysis, performing a crash landing dynamic analysis etc.).

For a given set of targeted downstream applications, i.e. for a given set of different FEAs and related objectives, we propose to study and clarify:

- The links between the data objects present in a DDMU, their organization and their consistency regarding the objectives of the FEA;
- The relationship between the DDMU at a given t time of its preparation process and the RDMU or the other related data sources, i.e. the links between the initial data sets and the adapted data sets;
- The way of generating such a DDMU, i.e. the DDMU preparation process.

10.2.2.1 DDMUs preparation process

A targeted downstream application (i.e. a FEA and related objectives) being specific, the preparation process that encompass both adaptation and/or enrichment mechanisms will also be specific and dependant of the requirements of the targeted application. To define and manage this DDMU preparation process we have been inspired by the DDMU preparation process proposed by [Drieux, 2006] and the concept of “Intermediate Model” defined by [Fine, 2001]. The approach consists in using a common and enriched product assembly model (DMU) as a support for the transformation of one design CAD model into one or several analysis CAD models used for FEA. However, according to what we have observed at Snecma and more generally in the aeronautics extended enterprise in terms of DMU usage, we have identified two different approaches for managing such DDMUs. Therefore we propose two scenarios that are illustrated in Figure 119.

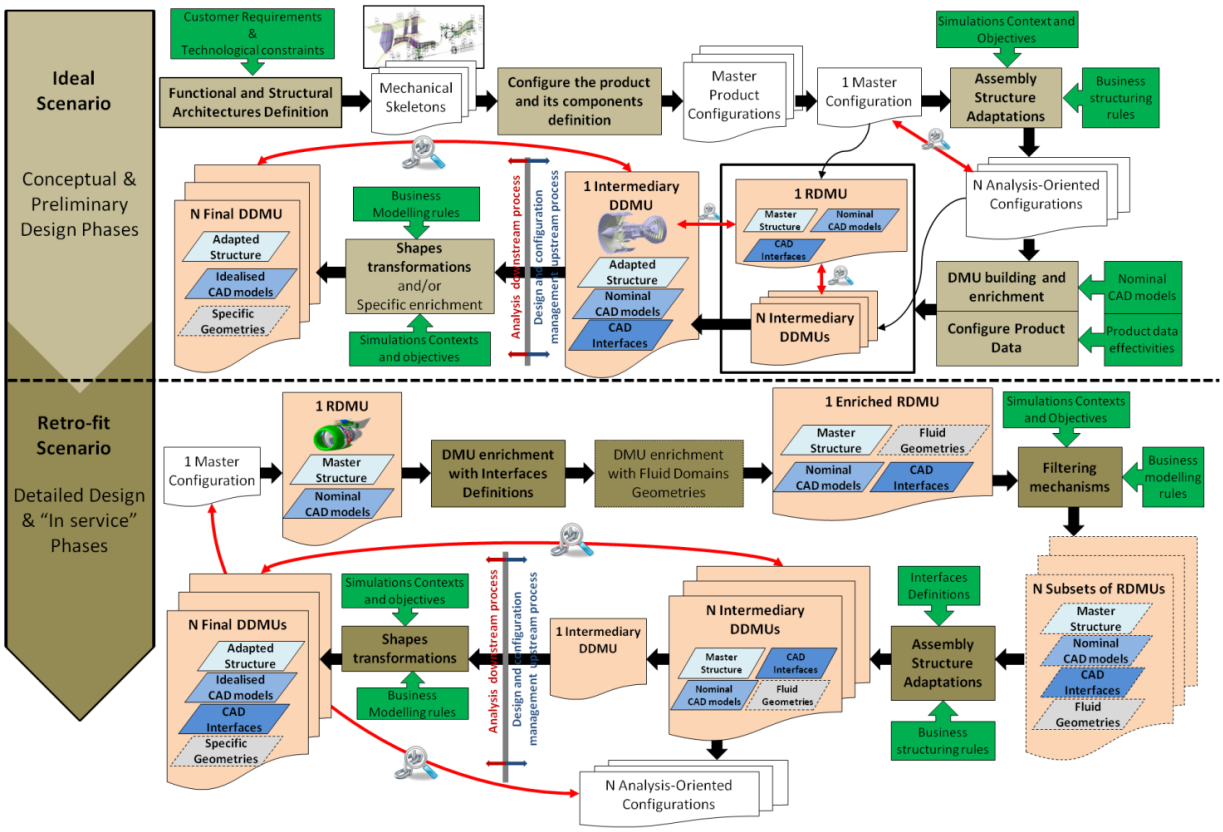


Figure 119: Two different scenarios and approaches for preparing and managing RDMUs and DDMUs

Scenario 1 or " Ideal scenario"- Enriching and managing one RDMU and related DDMUs product views at the same time from the earliest design stages

This scenario starts at the first stages of preliminary design when the DMU still does not exist. Then this scenario should be repeatable since the idea is to build, enrich and configure RDMUs and related DDMUs conjointly all along the development lifecycle. To achieve this objective, not only the designers and the configuration manager configure product data, but all the actors involved in the generation of DDMUs and their components definitions. This scenario starts from the first structural architectures leading to the first DMU representations: the mechanical skeletons (see section 3.3.3) that are still not configured. When these architectures are validated to launch the preliminary design phase, the master referential configurations can be defined. Based on pre-defined validation plan, and on the experience of previous projects, the derived "design discipline configurations" or "analysis-oriented configurations" can also be defined because designer already knows which kind of DDMU structure he needs for his specific needs. A master product configuration can lead to several "analysis-oriented configurations" so that their associativity must be ensured. Then, starting by using the context defined by the mechanical skeletons, the RDMUs and Intermediary DDMUs are enriched together: if a component is effective in a master product configuration and in a derived "analysis-oriented configurations", its definition (CAD models and properties) will be accessible in both configurations, and the design changes of this component (or its suppression for instance) will be propagated in both related product views. Since we have underlined the predominant role of interfaces definitions for FEAs, this DMU enrichment phase might also include the definition and specification of these interfaces all along design activities. Afterwards, from 1 one of these "Intermediary DDMUs", the analyst, according to the modelling requirements related to the FEA, can perform shapes transformation (mainly details removal and idealisations) to obtain a direct exploitable DDMU for the generation of an integrated FE

model compliant with the simulation objectives and related modelling requirements. The specific geometries in the final DDMU mentioned in Figure 119 concern for instance the fluid domain geometries used for CFD calculations.

Scenario 2 or "Retro-fit scenario" – Generating a DDMU from an already existing, complete, consistent and frozen RDMU

This scenario relates to the detailed design phase where DMUs are enriched with "as manufactured" parts and assemblies definitions. In this scenario, instead of enriching progressively the DMU along the development lifecycle with all required data and configure this data according to their relevancy for such or such applications, the DMU is enriched in "retro-fit" in view to the future adaptations. Starting from a selected "master product configuration" and its related DMU representation (a Referential DMU), a first step consists in enriching this RDMU with the definition of components interfaces (see section 10.3.3) and eventually enriching the DMU with fluid domains geometries (useful for aerothermal and CFD analyses). Afterwards, filtering mechanisms permit to filter the DMU content and create different DMU subsets regarding the required 3D geometries since not all components are modelled in 3D in the resulted FE model. Based on the interfaces relations defined previously, the structure of these DMU sub-sets are reorganised according to the business structuring rules and requirements of the FEA leading to a set of re-structured "Intermediary DDMUs" still containing the remaining (not filtered) nominal CAD models. These intermediary DDMUs will generate new product views that need to be considered as new "Analysis-oriented configurations". To obtain the direct exploitable DDMU for a specific application the shapes transformation process applied on these "Intermediary DDMUs" is similar to scenario 1 but still specific for the simulation objectives and related modelling requirements. This scenario is similar to the one proposed by [Drieux, 2006].

Currently this scenario 2 is usefull for on-going programs where already existing and consistent DMUs are available. It is not the ideal scenario we would recommand. However, in a "cross-project knowledge transfer strategy", it is mandatory to consider it since a new program can correspond to the development of a new product version that will be based on the definition and DMUs of its previous version. In this context optional steps can be added to this scenario including for example DMU scaling transformations.

10.2.2.2 Links between referential and downstream DMUs

In scenario 1, all DDMUs related to a RDMU evolve conjointly and the content and structure of the DDMUs are in principle always consistent regarding the RDMU. However there are two approaches considering a RDMU in a digital integration chain:

- Static frozen RDMU: A RDMU can be frozen for a given multi-disciplinary digital integration chain in order to ensure that all simulations involved in this integration chain are based on the same product definition. In that case, the related DDMUs are also frozen. When starting a new iteration, an update and refreeze of the RDMU and related DDMUs is necessary. This approach permit to avoid discrepancies between two independent asynchronous simulation loops (referring to the issue illustrated in Figure 10) in terms of used product definitions.
- Dynamic evolving RDMU: in that case the RDMU used in digital integration chain iteration is always evolving. The actors involved in the independent simulations are not obliged to use the same definition but might be able to retrieve which product definition have been used for such or such simulation models and results.

In both cases, whether a time or a knowledge gap is unavoidable. Indeed, the analyst involved in the fastest simulation loop that uses the results of the other simulation as inputs, will have to wait for the end of the other simulation if he wants to use consistent results. Otherwise, he has to use the most recent available results and be informed from which product definitions they have been generated. Thus, he might be able to compare the product definitions to analyse the design evolutions and their impacts on the simulation models and results. These gaps reinforce the importance of improving such a DDMU preparation process with the objective of reducing the time to prepare the simulation models.

In scenario 2, the DDMU preparation process requires a given time during which the RDMU evolves. Considering these evolutions, the creation of a simulation model for a given application can lead the analyst to execute some more or less regular and complete DDMU updates according to the type of design evolutions that have been integrated in the used configuration. These updates occur whether after the simulation loop to re-adjust the simulation models and results according to the RDMU up-to-date data, whether inside the loop to perform partial updates.

The update of the DDMU data inside or outside the simulation loop is hence sometimes necessary, justifying the implementation of adequate tools for change monitoring such as product configuration comparison and geometry comparison tools. Another approach, that we will further specify, consists in storing the nature, details and impacts of a design change within a “design change object” and to permit to the analyst to identify all the design change objects - impacting the used product configuration - that have been created between two time t.

10.2.2.3 Links between simulations and DMUs

The formalisation and capture of targeted downstream applications specifically dedicated to simulations and particularly FEA applications is mandatory for our approach. As already defined, such an application is defined by a **discipline** (or application domain), a **type of simulation** and a **specific analyzed behaviour** (the scenario), a **simulation context**, and some **simulation objectives**. Again, according to the nature of the downstream simulation, we have identified three different scenarios:

- New FEA: the targeted downstream application is new (the analysed behaviour, its type, its objectives, etc.) and/or never led to the creation of DDMU;
- Update of FEA: the targeted application and the corresponding intermediary DDMU (with nominal CAD models) and final DDMU(s) (with the idealized CAD models) already exist. However the DDMU used for the previous iteration has evolved and the analyst might be able to identify and analyse the design evolutions and their impact on the FEA models and results.
- Equivalent FEA: a similar targeted application (same discipline and same use case but different context) already led to the creation of a DDMU but for a different design and /or analysis data set (e.g. same simulation analyzing the same behaviour but analyzing a different assembly or product configuration)

The **simulation context** which gathers information about the **product lifecycle stage** (or program phase), the **system breakdown level** (or base-line), the **considered product configuration** is the object that makes the link between the simulation data set and the analyzed system or sub-system design data set (referenced by the system breakdown level and the product configuration) and representations; i.e. the DDMU in the case of assembly FEA if it already exists. The **simulation objectives**, pointing towards the design requirements of the analyzed system or sub-system that the simulation is supposed to verify/validate, permit to establish a link with the **functional definition** and the **design intent** of the analyzed artefact. Further in this chapter we propose to use the concept of simulation intent to capture

the simulation objectives with associated modelling requirements. Therefore the data model supporting such concepts might gather all these study context information. Figure 120 shows a **study management** package which is composed of several classes that permit to capture the required information for defining a study context.

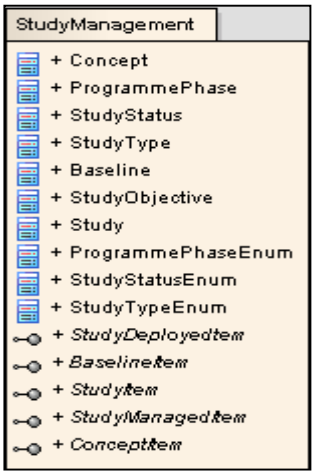


Figure 120: Content of "StudyManagement" package

The class **Study** might be the main entry to collect and access to the different context of the study inside the design process. It is a central class that gathers all the information about the study. The other class ensures the definition of the context of the study:

- **Concept / Study Objective / Study Type:** the user might define the goal or simulation intent (design requirements to verify or concept to experiment), the type of simulation to perform (mechanical, thermal, linear, non-linear, etc.) and the methodology to do it;
- The **Baseline** gathers all the data linked to the product and its configuration and the models associated to it;
- The **Programme Phase** indicates the stage of the product development process in which the simulation is performed (Feasibility, Conceptual Design, Detailed Design, Certification).

All these classes that are used to describe the study context might be associated to other packages in order to access to all the data required for performing the study: product design definition, models, boundary conditions and load cases.

10.2.3 Conditions and required capabilities for exploiting downstream DMUs in design-simulation loops

In this section we define the list of the conditions and required capabilities for exploiting downstream DMUs in design-simulation loops and for considering the DMU as a flexible product definition reference for handling collaborative digital integration chains. This is essential to identify the entire scope of the development phase and for prioritising the development tasks regarding the potential impact of these capabilities, the effort required to implement them and regarding the scientific positioning of this PhD. These conditions and capabilities have been classified according to their functional use cases.

In terms of DMUs consistency with product definition and other product representations:

- *C0a: Starting the DDMU preparation process with a complete consistent RDMU (prerequisite for scenario 2);*

- *C0b: Possibility to use a "3D-2D operator" permitting to derive from a RDMU or a DDMU 2D cross-sections used for instance for 2D axisymmetric simulation models (prerequisite for ensuring the consistency of the final integrated FE model);*
- *C0c: DMU enriched with material references to be able to derive the mass properties (prerequisite for deriving BOMs from DMUs and ensuring their a DMU-BOM consistency);*

In terms of product configuration management process capabilities:

- C1: Possibility to configure the product according to design/analysis discipline-oriented product structures;
- C2: Possibility to integrate new kind of product data (such as idealised models, interfaces definitions, fluid domains, behavioural item definitions and related models) within these configurations;
- C3: Consistency between master product configurations and analysis-oriented configurations;
- C4: Possibility to use design change monitoring and analysis capabilities such as a product configuration/structure comparison tool or geometries comparison tools;
- C5: Possibility to capture inter-related design change within a "Design Change Object" and retrieve these design changes between two instants t.

In terms of DMU enrichment mechanisms:

- C6: DMU enriched with interfaces definition integrating the functional, topological, and technological aspects of the interface;
- C7: Possibility to use operators capturing topological contact areas between components (mainly for scenario 2);
- C8: Possibility to create and re-use interface templates in order to easily integrate these interface definitions within DMUs;
- C9: Possibility to use operators to extract fluid domain geometries and to consistently integrate and configure them within the DMU;
- C10: Possibility to define/specify structural and behavioural architectures and to enrich/adapt DMUs jointly and consistently with these architectures;

In terms of DMU adaptation mechanisms:

- C11: Possibility to use a "3D-System" operator permitting to derive from a DMU assembly and related interfaces definitions the corresponding multi-level system model representing the components, their interactions and permitting to navigate through the system breakdown levels;
- C12: Possibility to declare business structuring rules on these system models and specific interface definitions to re-organise DMU structures;
- C13: Possibility to filter a DMU according to simulation objectives and related modelling requirements;
- C14: Possibility to use idealisation tools and to monitor the necessary idealisations according to the simulation objectives and related modelling requirements;

In terms of links between DMUs and simulation data and meta-data:

- C15: Possibility to reference a product base-line and a product configuration in simulation templates to retrieve which product definition has been used for such or such simulation models and/or results;

- C16: CAD-CAE data high level associativity to retrieve which CAD model has been used for such or such simulation models and/or results;
- C17: CAD-FE topological associativity to identify the relevant topological elements in both CAD and FE models in order to associate an identified interface CAD mating feature to the corresponding FE mating feature or to apply the relevant boundary conditions on the right FE elements;
- C18: Possibility to capture a DDMU preparation process and associate it to a specific targeted downstream application (FEA in our case) and to re-use it on updated (in the case of a FEA update) or different data sets (same kind of FEA applied to a different context).

In terms of interoperability and collaboration:

- C19: Possibility to visualise and modify a DMU in one or several CAD modellers or viewers;
- C20: Possibility to exploit a final DDMU in the right FE pre-processing environments;
- C21: Possibility to exploit definitions of behavioural architectures (defining the structure and content of an integrated FE model) in the right FE pre-processing environments in order to automate the assembly of FE models;
- C22: Possibility to exchange a product definition (and particularly structural and behavioural architectures integrating interfaces design, behavioural definitions and modelling specifications) between heterogeneous PDM/SDM systems.

10.3 DASIF related capabilities and concepts proposal

Considering all these conditions, the scope is too large to integrate the development of all related required capabilities in the scope of this PhD. Therefore we have identified the priorities and this research work focuses on:

- Specifying the capabilities required to respect the pre-required hypotheses (C0a, C0b, and C0c);
- Specifying some transformation and enrichment mechanism and related operators (C6, C7, C8, C9, C11, C12, C13);
- Defining the DASIF product data model supporting such a use of DMUs, supporting the system integration framework and design-analysis data integration (C1, C2, C3, C5, C15, C16, C17);
- Specifying and develop a system integration framework using an object-oriented system modelling language to define/specify multi-level and multi-domain system architectures, to enrich and adapt DMUs conjointly and consistently with these architectures and to support design-analysis data integration (C10);
- Studying how to exploit behavioural architectures definitions FE pre-processing environments in order to automate the assembly of FE models (C21);
- Supporting standardized exchange of product and simulation data within collaborative aeronautics digital integration chain and hence between heterogeneous PDM-SDM applications by mapping these concepts and the proposed data model to standardised product data models (C22);

The solutions and concepts proposed in this section are based on three important hypotheses:

1) Imported product definitions and representations are considered consistent;
2) Existing DMUs and related CAD models are enriched with mass properties (mass, centre of mass and moments of inertia) and a material reference attribute;

3) The product is developed in a collaborative and distributed design environment where sub-systems are designed at each partner site.

10.3.1 Referential DMUs and derived representations

The first capabilities to develop must ensure that the previous mentioned hypotheses are fulfilled. In order to get a DMU consistent with the current product definition and other related representations, we propose to use the DMU as the reference for generating other representations of product definition such as the Bill of Materials and the 2D drawings.

10.3.1.1 Integration of mass properties in DMUs to generate BOMS

A Bill of Material is a list of all individual parts constituting the assembly. For each of these parts, the BOM provides the following information:

- The designation of the part;
- The part number: code referencing the item;
- Its functional reference (SNS in the aeronautical industry): code referencing the function of the part in the assembly;
- The quantity: when the same part is multi-instantiated within the same assembly;
- The material reference;
- The mass properties of the part: the indicative mass, the coordinates of the centre of mass and the inertia moments.

In order to generate a BOM from a DMU, it is therefore required to integrate all these information within the DMU. As-is DMU already contains the name, the part number and the functional reference of its components since they are already stored and accessible in the PDM data base. However some current DMUs still do not consider the mass properties of the components and the material reference of the parts. In PDM systems, the definition of the system components is generally structured around these three entities:

- The definition of the item: which is common to all the instances of this item (designation, part number, CAD model, etc.);
- The definition of the item instance: describing features related to the instance itself (function and related functional reference, placement in the parent assembly) and describing the children components of the item instance if the item is an assembly;
- The product structure link or node (or assembly_usage_occurence): describing the attachment of an item instance to the definition of its parent assembly. This entity can contain the same information related to the item instance since there is one structure node for exactly one instance.

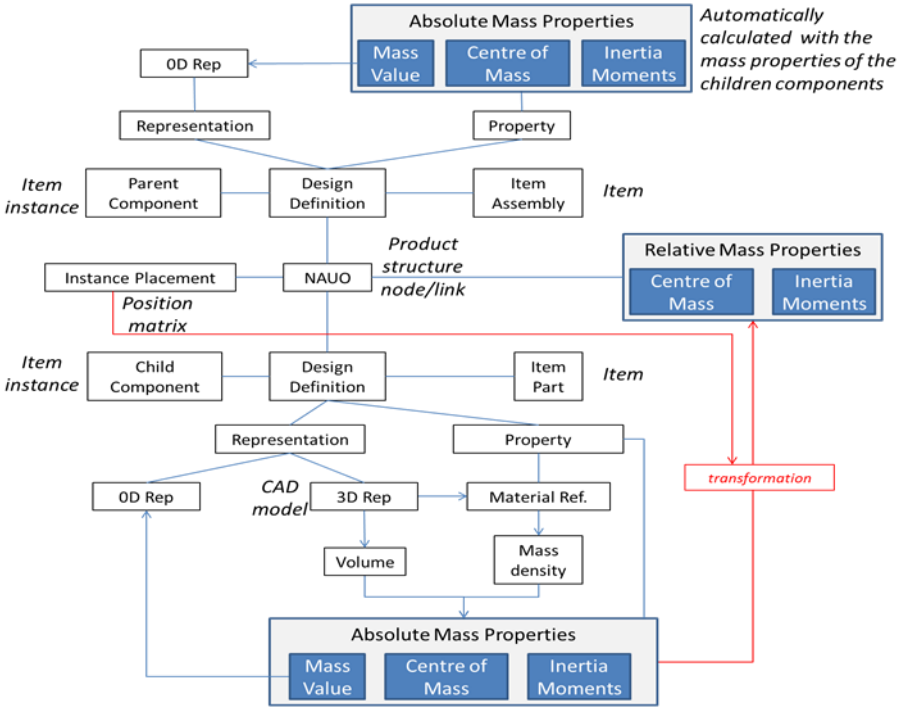


Figure 121: Mass properties management principle proposal

In order to integrate the mass properties and the material reference within DMUs, we propose to integrate them in the CAD model parameters and we recommend setting a DMU rule: “A component can be integrated in the DMU if and only if its CAD model contains a material reference enabling to derive the absolute mass properties”. Figure 121 provides the conceptual model we propose to support this capability. Moreover, the mass property must be an alternative to the 3D geometrical representation (representation "0D"). This representation must be attached, such as 3D representation, to the definition of the part item. Services must be able to derive the mass properties of a system from the mass properties of its components.

10.3.1.2 Generation of relevant 2D drawings from DMUs

For products which geometry is mainly axisymmetric, analysts performing simulations at the top integration level often use 2D axisymmetric models in addition to 3D models to accelerate the computation time. In that case the main design data input is not a 3D DMU but a 2D drawing representing a cross-section of the DMU. Currently the cross sections used for 2D axisymmetric models present some inconsistencies with the corresponding DMU:

- Inconsistencies related to the delta or gap between the current technical product definition and the status of the DMU: because the creation of 2D cross-sections is not synchronised with the DMU evolutions (see Figure 122).
- Inconsistencies related to the fact that the cross-section does not include certain non-axisymmetric elements that need to be considered and/or modelled for the simulation
- Topological inconsistencies: large cross-sections often have some non-closed outlines forcing the analyst to manually refine the corresponding edges.

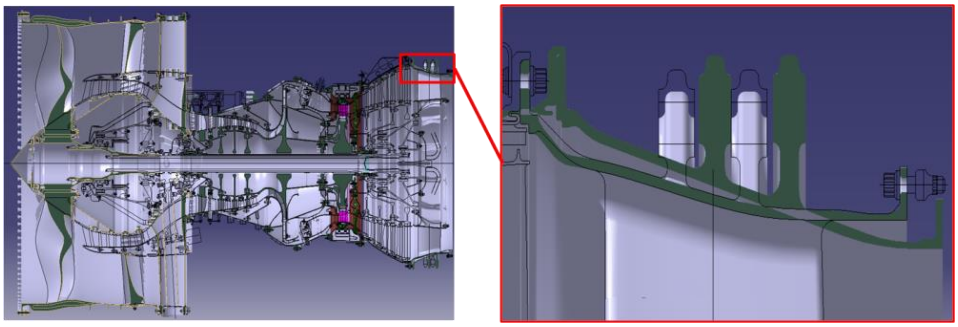


Figure 122: example of 3D-2D inconsistencies between a DMU and a 2D cross-section

To avoid these inconsistencies we propose:

- To generate these cross-sections directly from the DMU and to capture the link between the 3D CAD assembly model and the equivalent 2D CAD cross-section (associating these models to the same definition of the assembly).
- To integrate in the cross-section the relevant required non-axisymmetric elements regarding the simulation objectives: these elements must be projected on the cross-section plan.

When considering the non-axisymmetric elements in the simulation model, mechanical or thermo-mechanical analysts need the information concerning the "filling ratio" of these elements. The "filling ratio" is the ratio between the volume currently occupied by the non-axisymmetric element on the total circumference and the volume occupied if we would perform a complete revolution of the same element on the circumference. For an axisymmetric element the ratio equal to 100%. For instance, the impact of ventilation holes for a rotor part with a small diameter may be significant for mechanical analyses and should be taken into account in the 2D axisymmetric mesh used for stiffness calculations. Therefore, this "filling ratio" information must be accessible directly from the 2D cross-section.

Finally associativity between 2D topological elements of the cross-section and 3D topological elements of the 3D DMU is required so that:

- The 2D cross-section can follow automatically the updates of the DMU;
- There exists a bi-directional associative link between the data sets used for 2D and 3D computations;
- This associativity can be exploited in pre-processing environment.

Figure 123 shows the proposed methodology and related process to provide these consistent 2D cross-sections.

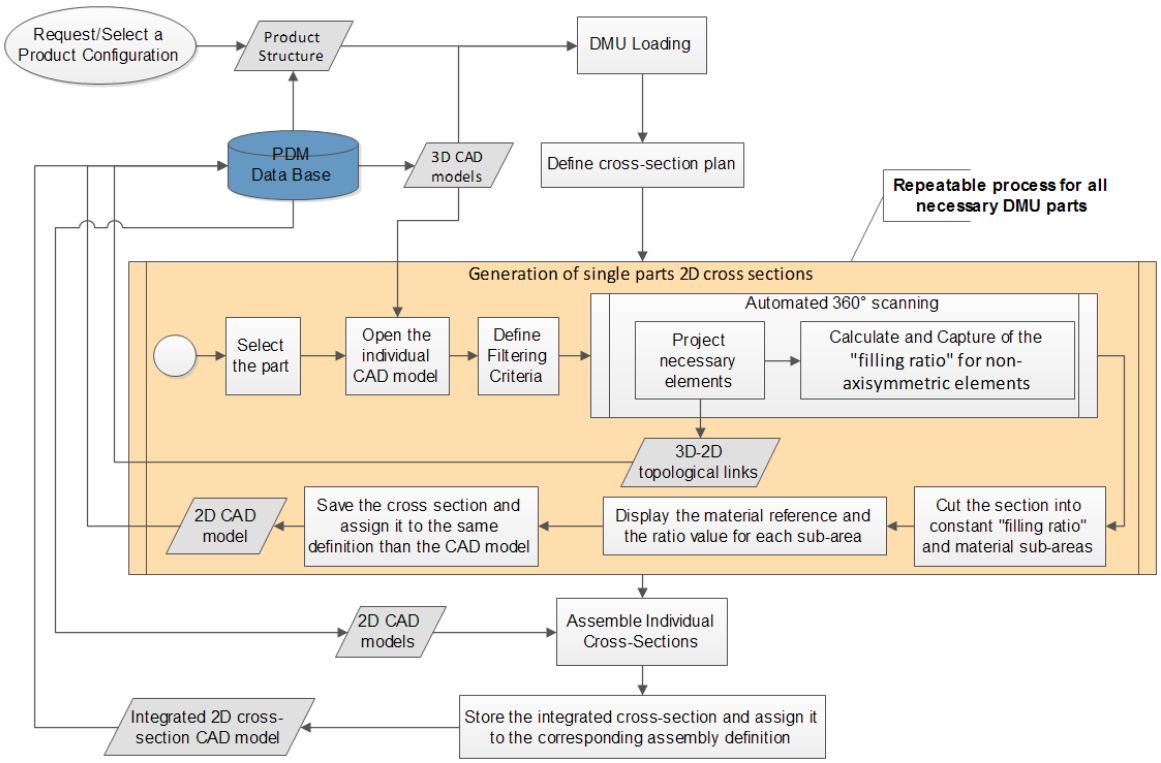


Figure 123: Proposed To-Be process for generating DMU-based and simulation-oriented 2D cross-sections

This process is made of the following steps:

- DMU loading and definition of the cross-section plan: the process starts by requesting and retrieving the desired product configuration in the PDM client application and to load the corresponding DMU in the CAD modeller.
- Generation of single parts 2D cross sections: this step is the main step of the process and is a sub-process that might be performed for each DMU parts that need to be projected in the cross-section: the user only perform this sequence for the relevant parts that might appear in the cross-section. This sub-process is composed of the following steps:
  - **Select a part and open the corresponding CAD model**;
  - **Define filtering criteria**: the filtering criteria are defined by the user and are based on commonly used CAD features (such as holes, embossing, chamfers, etc.). The user might select in a feature list, the ones that will not be projected in the cross-section;
  - **Projection of the necessary elements and generation of the cross-section**: the tool might perform a 360° revolution scanning (around the axis orthogonal to the cross-section plan) of the part geometry and project all the related topological elements (point, edges and curves) on the cross-section plan, except the ones corresponding to the CAD features selected in the previous step. This is during this step that the tool might automatically capture the associative links between the 3D topological elements and the 2D topological elements that it has generated;
  - **Calculate and capture the filling ratio of non-axisymmetric elements**: while scanning the geometry, the tool might be able to calculate the range of degrees where there is no “solid matter” and derive the corresponding “filling ratio”. If this ratio is not constant for the whole part geometry, different ratios might be assigned to different geometric areas according to an adjustable “precision parameter” defining for which “ratio step” the tool generate a new geometric sub-area;

- **Cut the section into constant "filling ratio" and material sub-areas**: based on a material parameter (defined for all parts or for each body of the part if the part is made of several materials) inherent to the CAD model and on the "filling ratio" measure of the previous step. This step consists in providing a visible cutting of the section that gather the geometric areas where the material is the same and where the "filling ratio" is constant (the expected result is illustrated in Figure 124);

➢ Assembly of individual cross-sections: once the cross-sections of all necessary DMU parts have been generated, the tool might be able to progressively assemble all the individual cross-sections of the DMU parts. This assembly is only based on the relative position of each part in the reference frame of their parent assembly.

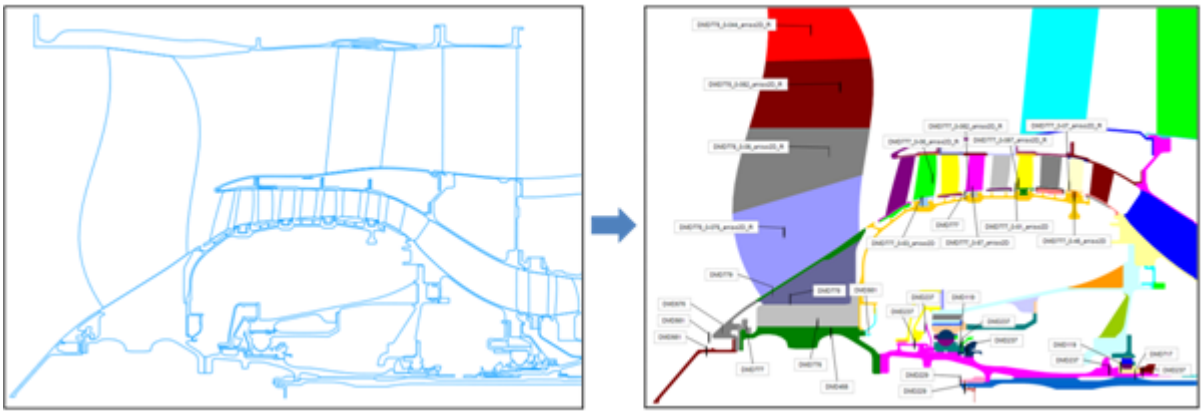


Figure 124: illustration of the cross-section cutting results on a turbojet FAN module

Recommendation: in order to generate a coherent integrated cross-section, the filtering criteria and the "ratio precision" parameter might be the same for all the assembled parts.

10.3.2 Multi-view DMU Configurations and Product Views Management

10.3.2.1 Main product configuration principle

The definition of the components (item instances) that constitute a configuration is managed through the concept of "effectivity". An effectivity is the identification of a domain of applicability for product data [ISO, 2004a]. From a PDM/PLM perspective, the effectivity is a product structure link attribute that defines if a usage occurrence of an item definition (i.e. instance of a system artefact) is relevant for a given configuration or list of configurations. The principle of product configuration applying effectivities on artefacts "Definition_Usage_Occurences" is illustrated in Figure 125. This principle is based on classical PDM and standards recommended practices. There might be different kind of effectivities as already illustrated in Figure 75. One effectivity can be a list of configurations, a range of configurations (if they are sequentially defined) or even a dated_effectivity (with a start date and an end date; this supposes to assign a start and an end date attribute on the configuration entity).

Each product is associated to a "Root" entity which is the top-level component the non-configured product structure and carrying the configurations for design and manufacturing. It might be noticed that it should be possible to define non-configurable structure nodes if it agreed that one component might be present in all configurations. It could be the case for the highest level components.

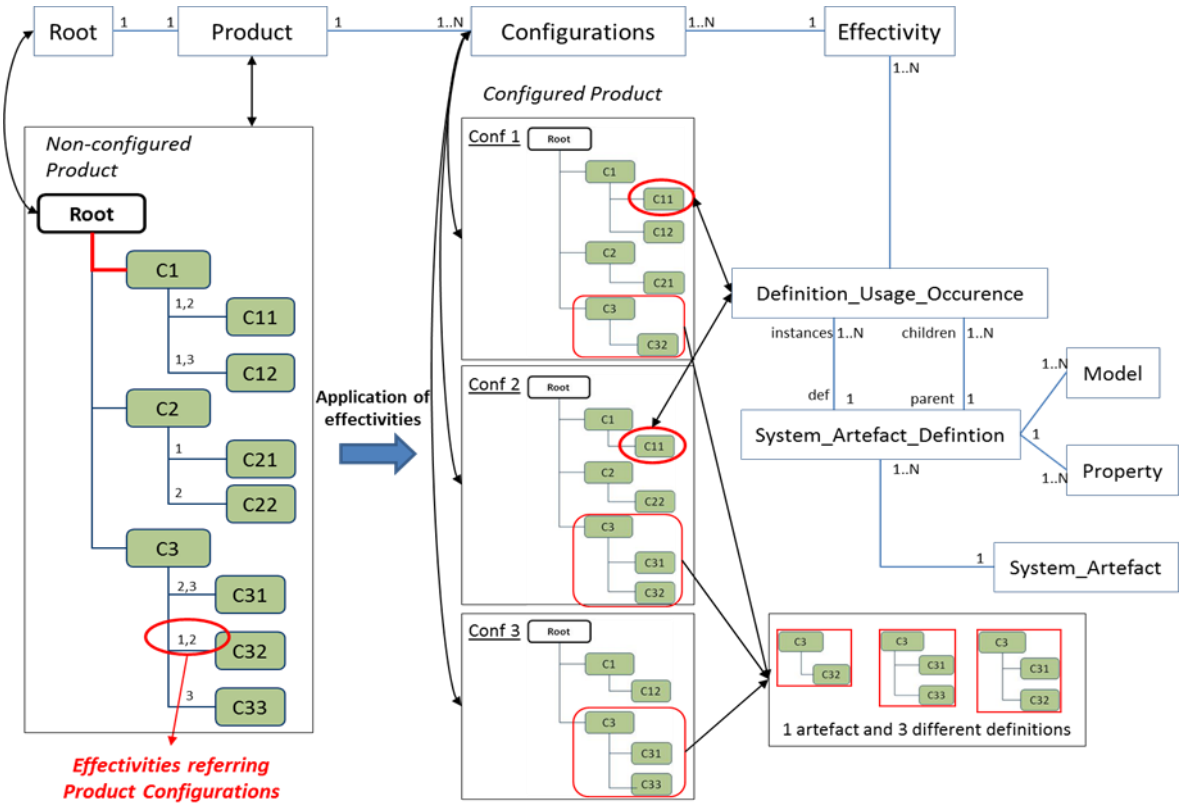


Figure 125: Effectivity-based product configuration working principle

10.3.2.2 Master, Manufactured and Design Discipline Configurations

A master product configuration is established each time it is necessary to agree on a reference which is the basis for identifying further configurations and related product views. In current configuration management practices, they are commonly three main types of master or referential product configurations according to the development cycle stages [BNAE, 2003]:

- Functional referential configuration: is a reference agreed for launching the preliminary design phase. It generally corresponds to the first described functional architecture (structure of functional packages/modules and related design requirements) and/or to the first conceptual and structural architectures.
- Development or Design referential configuration: is a reference agreed for launching the detailed design phase. These configurations take into consideration the items identified and designed at this stage of the development process. An item configuration at this stage is generally composed of its design requirements and of the technical definition elements fulfilling these requirements (i.e. the structural architecture of the item, its preliminary internal interfaces definitions, performances allocations, its geometry and other agreed definition elements like the parts materials)
- Manufacturing referential configuration: is a reference agreed for launching the industrialisation/manufacturing phase. Such configuration is achieved when the technical definition of its constituting items and their integration have been validated and enough justified to serve as reference for manufacturing and assembling these items.

The traceability and consistency between these configurations must be ensured. As illustrated on Figure 126, further configurations based on these master configurations, and particularly the ones defined for specifying given design or analysis views of the product, are required.

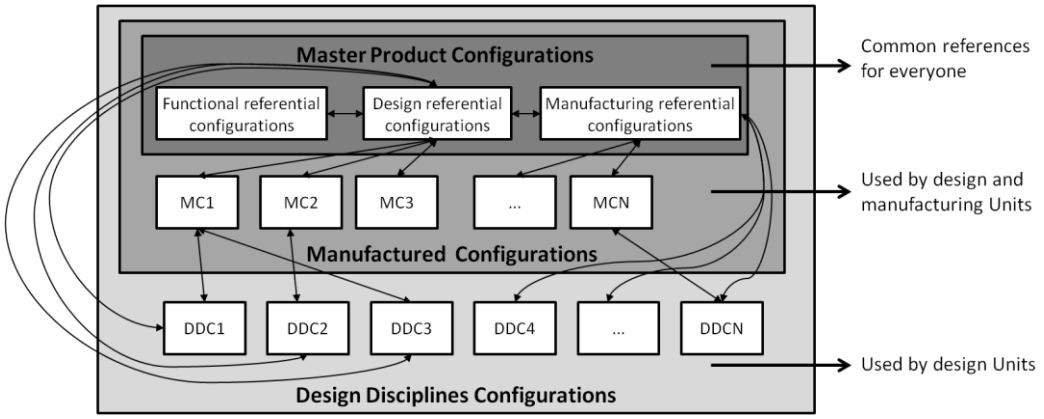


Figure 126: Master, Manufactured and Design Disciplines Configurations

Reminding that a view is a restriction (or subset) and a specific data organization of a product defined to provide an understandable and appropriate representation according to a specific business perspective, one approach could consist in defining a design discipline configuration each time a specific view of a product is required. However, with such an approach the configuration management process and the assignment of product component definitions in the relevant configurations can become complex. A view is hence defined by a re-organisation of the product structure and by filtering the relevant elements that appear in the view. In STEP, "the View_definition_context" entity defines the context in which the design definition of an artefact (properties, documents, structures, etc) is valid. A View_definition_context is the grouping of an application domain and a life cycle stage. Therefore we propose an extension of this vision as illustrated in the UML class diagram in Figure 127.

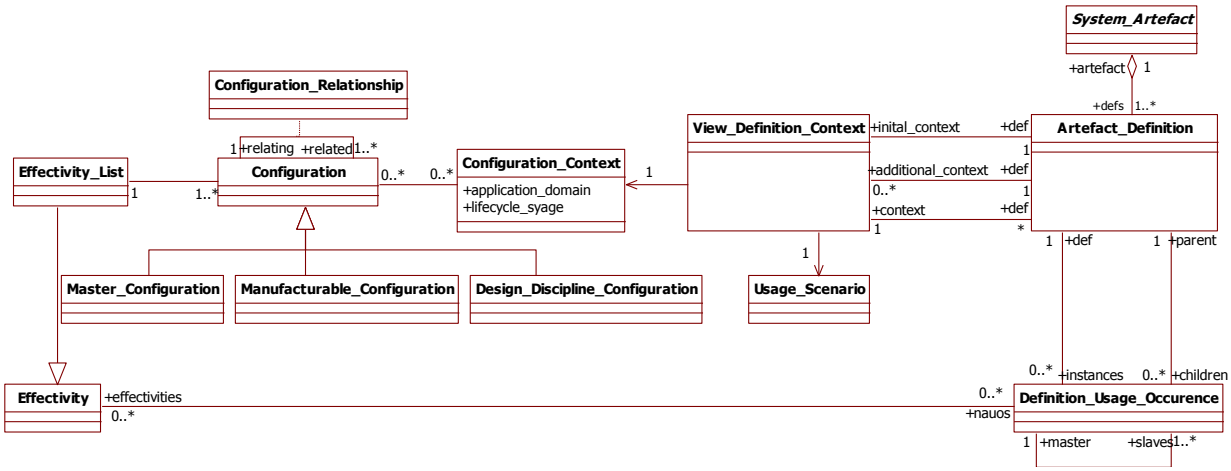


Figure 127: Conceptual model for managing configurations and related product views

Configurations are defined in a specific application context specifying the application domain or discipline and the product development lifecycle stage for which the configuration is relevant. A view being associated to a specific configuration is associated to this context. However, a "view definition context" needs additional information justifying the specific used representations: the usage scenario. In the context of this PhD, the usage scenarios will be the different FEA simulations using this view.

10.3.2.3 DMU views associated to configurations

In the context of configured DMU product representations, a view is also obtained, according to the objective of the targeted application, by filtering the DMU content (selecting only relevant parts) or/and by the use of specific representations for the components present in the configuration. For

instance, instead of filtering a component 3D model, it could be replaced it by its equivalent 2D cross-section or by its 0D mass representation. Therefore, for a same design_discipline_configuration several DMU product views can be defined. These DMU views have the same structure between assemblies, sub-assemblies and parts, but display different items representations and hence specific items definitions. A DMU view can also be a final idealised and exploitable DDMU for FEA if the idealised representations are available. These DDMU configurations and views concepts and the way they are managed are illustrated in below Figure 128.

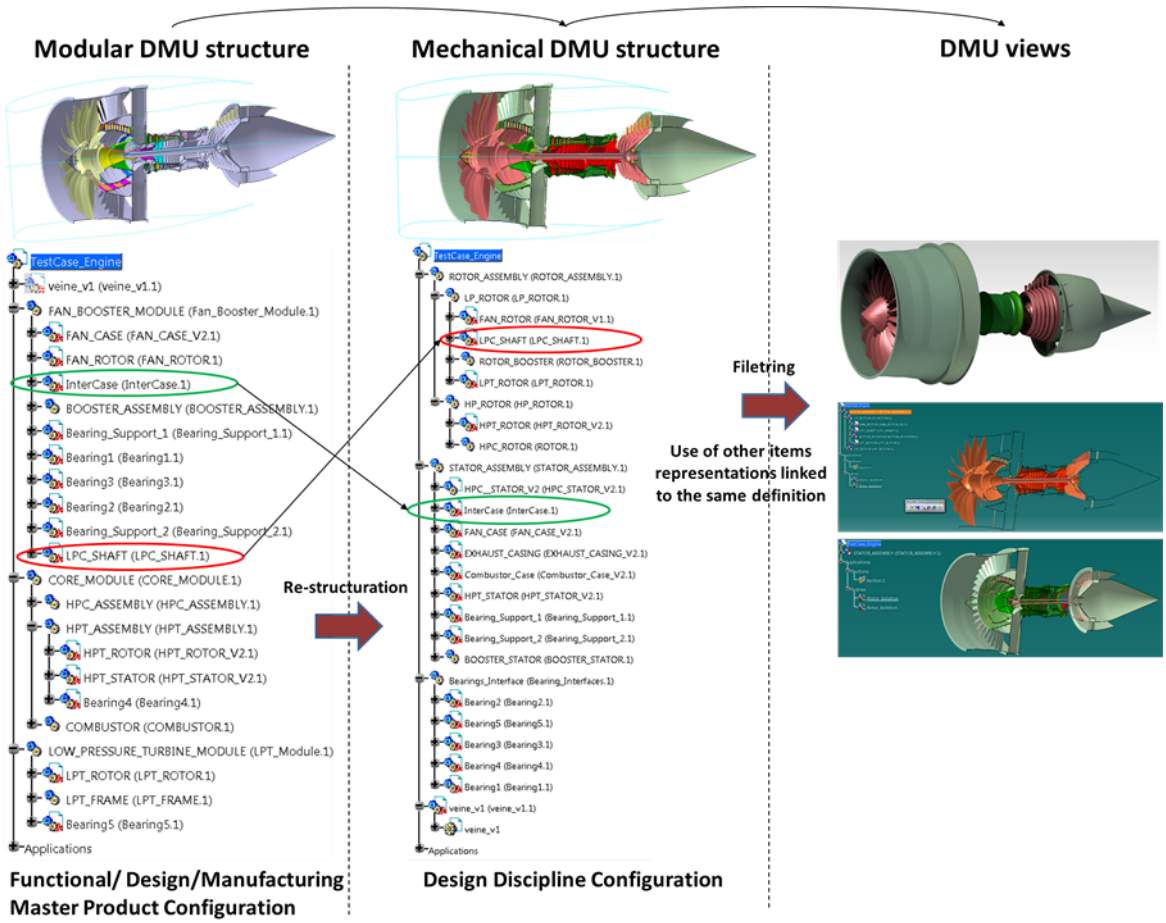


Figure 128: Illustration of a multiple usage occurrences of DMU components in a modular referential configuration, in the mechanical configuration and some possible derived DMU views.

Therefore, an artefact definition might be associated to a specific set of representations. The view_definition_context will allow capturing for which context(s) and usage scenario(s) this definition is relevant and used. For instance the definition of a fluid domain and its associated geometry will only be used in specific thermo-mechanical and aero-thermal usage scenarios (CFD calculations).

10.3.2.4 Engineering change propagation between DMU product views

As mentioned in section 4.1.3.2, engineering changes can be a major source of inconsistency if they are not properly propagated within the various product configurations and related representations. Therefore, companies require product data views for each domain that support their specific views and EC propagation procedures that maintain consistent product data between design departments. Figure 129 proposes an engineering change process schema relating the engineering change process to the design change objects and the product data definitions. In this schema, the process should be monitored by a review workflow aiming at defining all the features and impacts of the design

change. The design change makes evolve the item definitions in different ways (CAD modification, structural change, deleted component, property change, etc.). The design change can be propagated in all configurations where the impacted item definitions are effective.

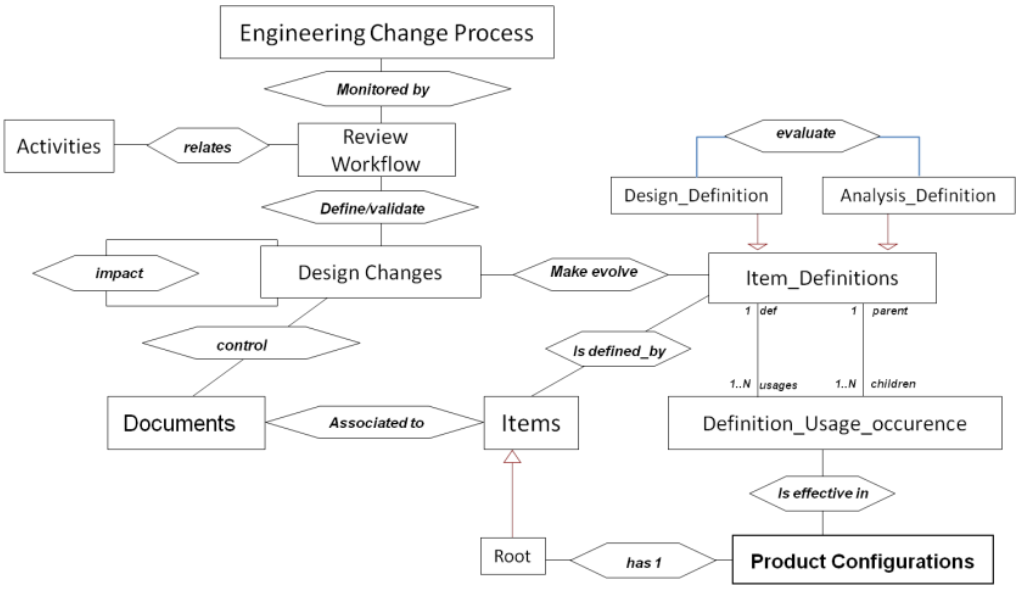


Figure 129: Engineering change process schema

When a design change occurs, such as a modification of one of the DMU components geometry, a new version or a new instance of the artefact_defintion is created. The previous definition had potentially several definition_usage_occurrences effective is several related master and design discipline configurations. Therefore a mechanism is required for propagating the changes in these configurations to maintain them up to date.

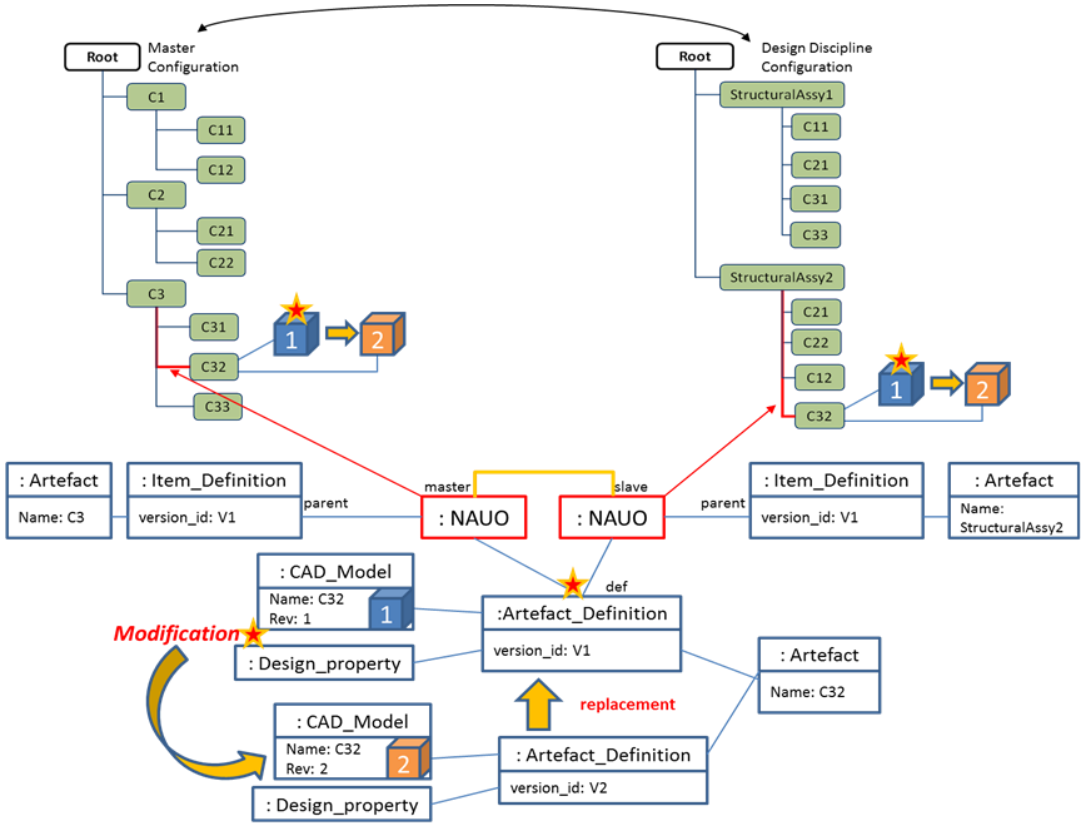


Figure 130: Change propagation from a master configuration to a related design discipline configuration

The mechanism we propose is illustrated in Figure 130. The propagation relies on two important relationships: a master-slave relationship relating the two configurations and a master-slave relationship relating the two design_usage_occurences (NAUOs). The propagation might occur only when the design change is validated. The related design change object might have a status attribute permitting to trigger or not the propagation. When the CAD modification is validated the previous Artefact_definition is replaced by the new one. The new artefact_definition might retrieve all the definition_usage_occurences which was associated to this definition in order to assign the new definition to all relevant configurations. The master-slave NAUO relationship ensures for instance that when a component (master NAUO) is deleted from a master product structure it is also deleted on the other configurations. If new components appear in a master product structure, the slaves NAUO has to be manually created. In that case, the definition of the parent component has changed (since it has a new child component) and if the parent component definition is effective in other configurations, the modification will be propagated as soon as it is validated with the same mechanisms. There are many possible scenarios for monitoring the engineering change propagation process. These scenarios highly depend on the nature of the change and on the impact of the change on other product components. These different types of change and related scenarios are further explained in the next chapter while specifying the conceptual data model.

10.3.3 DMU enrichment with digital interfaces definitions

10.3.3.1 Interface types

The interface is a real or virtual area where there exists an interaction between two elements. In this area, the interaction enables to link two elements to ensure functional, structural and behavioural continuity. There exist several types of system's interactions. Some interactions are functional (intended) and the system behaviour and performances highly depend on them. Other non-functional interactions can be identified and specified in order to better simulate the unintended phenomena and anticipate them. In the context of DMUs, whether the interaction is functional or not, we propose to consider three types of interfaces:

- "Solid-Solid" interfaces: it is an interaction or non-interaction (clearances) between two physical components. Except for clearances, which will be processed separately, these interactions occur between geometric features of the components (contact between two surfaces, for example). They are characterized by a kinematic and the resulting degrees of freedom which are needed to model the interface behaviour. For an aero-engine, the two most commonly used interfaces which define the overall design context are the bolted flanges and the bearings. However we can mention other sub-types of mechanical interfaces: engine-equipments interfaces, suspensions interfaces, groove interfaces, etc.

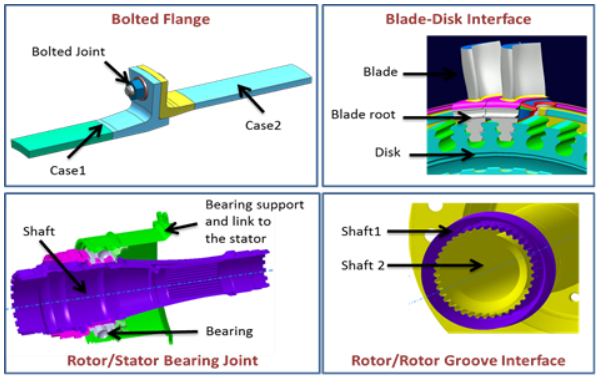


Figure 131: Examples of solid-solid interfaces usually found in a turbojet engine system

- "Fluid-Solid" interfaces: It is any kind of interaction between a fluid domain and physical components of the product (the solid domain). They can be generally classified into two sub-categories:
  - Pressure of a fluid on a solid surface: to consider the displacements of the solid surface resulting from the applied pressure.
  - Thermal/Heat transfer on a solid surface: this type of interaction need to be analysed to consider the solid displacements of the solid surface resulting from the heat transfer (modifications of the material properties)

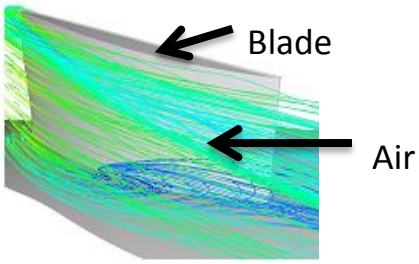


Figure 132: Example of fluid-solid interface

- "Fluid-Fluid" interfaces: These are all the other interactions occurring between the product fluid domains. They can be materialized physically by ventilations holes or scoops. These interfaces are necessary to specify for instance the section of a functional air intake designed for cooling down a hot area in the "high pressure" parts of the engine.

All these interfaces are characterized by a function (the role of the interface and the related design intent), a structure (localisation and geometrical definition of the interface) and behaviour (the behavioural representation of the interface and its related parameters). This characterization is not dependant on the type of interface. The type of interface will impact the type of representation used to describe the interface as well as the associated modelling parameters.

10.3.3.2 Creation of an interface

When the product is designed, the different pieces that composed the model are created and defined in the way to ensure the function that they have to perform. It is the same for the interface. The designer create interface to ensure a specific function. An interface is always defined as a way to ensure the link between two components and the continuity of the mater, and supports the flow of energy that goes through one component to another. The design of interface starts with the research of a technical solution. The choice of the solution is constraints by some technical specification that are:

- The **geometric constraints** of the interface which are given by:
  - The shape of the surface of each component in interaction
  - The location of the interface inside the product
- The specification regarding the **function to perform** (Strength, Temperature Pressure, Weight...)

In this situation, the designer, even before any simulation, knows the type of behaviour that the interface must have – i.e. the **design intent**. The designer defines an interface which fits with the requirements of the function to ensure. This information is created and might be captured from the creation of components' geometry. The interacting parts are then identified and specified for the assembly. Instead of specifying this information via Interface Control Drawing documents (as-is practice) we propose to integrate all these information within a DMU through the use of digital interface objects that might gather the following information:

- The function: defining the role of the interface within the system (e.g. "to make a rigid connection between the two component", "to ensure a thermal exchange between two fluid domains"...)

- The design intent: justifying the choice of the type of interface;
- Application domain (mechanical, thermal, etc.);
- Type of connection (the kinematic pair for mechanical joints, convection/conduction for thermal interfaces, etc.)
- Associated technological components ensuring the interaction (bolted flange, bearing, etc.) and their representation (e.g. CAD model of bearing or the bolt-nut assembly).
- Identification of the parts and geometric elements that interact;
- Assembly constraints and mating features at these interactions (contact, coincidences, clearance, plot, etc.);

This description need to be directly accessible from the DMU by selecting the component that manages the interface. Information listed previously is associated to the geometric data (CAD files) used to represent the interface, and managed inside the DMU. The DMU ensures the update of the definition of the interface but also to manage different design alternative for the connection. Moreover, the DMU enables to share the information about the interface among the different design departments.

10.3.3.3 Capturing product component relationships and CAD mating features within DMUs

We propose to integrate interface in product structures and DMUs as a specific type of product components (new kind of system artefact) in order to access all these data sets. When integrated within a product structure the “interface component” must be placed in the tree structure at the same breakdown level than the inter-related components. This position in the tree structure hence depends of the used DMU product view/configuration in which the interfaces are specified. This is illustrated on where bearings interfaces are defined at the same level than the stator and rotor assembly components, whereas in the modular reference structure they are spread in their respective modules.

Information listed previously is associated to the geometric data (CAD files) used to represent the interface, and managed inside the DMU. The DMU ensures the update of the definition of the interface but also to manage different design alternative for the connection. Moreover, the DMU enables to share the information about the interface among the different design units.

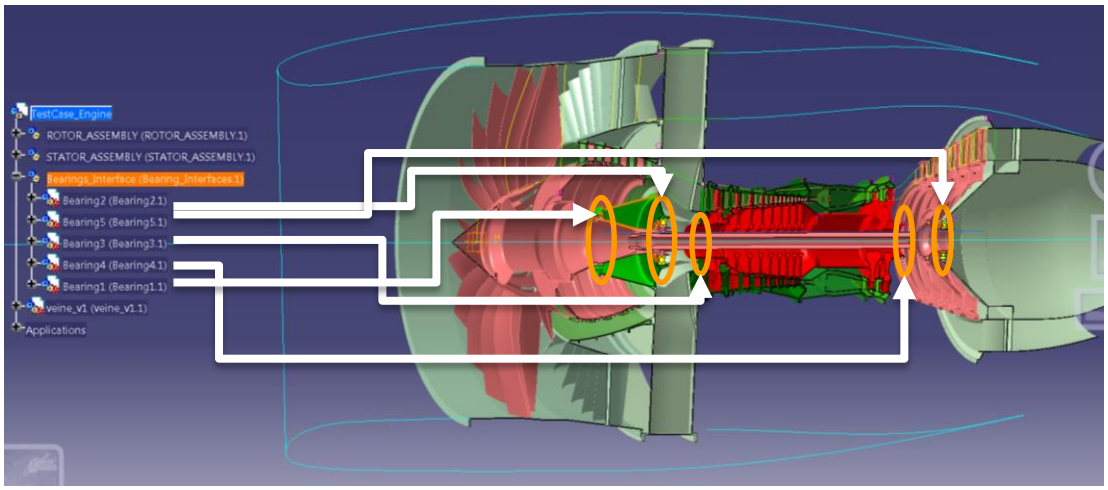


Figure 133: place of the bearing interfaces in a mechanical DMU structure

The topological (identification of mating faces/features) and technological definition (technological components) of the interface can be accessed directly using the CAD models. However, the topological definition will be accessible only if it has been captured previously. We have identified two

possible scenarios for capturing the design intent, the topological and technological definition of interfaces. These scenarios correspond to the two scenarios mentioned in 10.2.2.

1) Scenario 1: Specifying the interfaces during CAD design modelling

In this case the topology of the interface (mating features) cannot be defined directly since the geometry of the inter-connected components is still not defined. However, in preliminary design, one of the first design steps consists in building a product structure according to the pre-chosen architecture and defining the interfaces between modules. Then the internal module interfaces are also defined to permit the module designers to start designing their parts in context. For an aero-engine, the first interfaces to be specified are the “inter” and intra-modules bolted flanges and the bearing joints between the rotor and the stator. As a result the first mating features are explicitly defined (e.g. plane and alignment of bolted flange) and must be captured. To achieve this we propose to use **CAD interface templates**.

CAD interface templates are geometric entities created or instantiated in the CAD modeller (in our case CATIA V5/V6) via the "Product Knowledge Template" application in order to create and capture the design context. The templates are created either directly or created from existing geometric entities in other CAD models and are called instantiation. They can contain not only geometry, but also all the associated parameters or relationships, including design rules, providing the ability to encapsulate the design specifications for the design context. For their positioning, the **CAD interface templates require** input data such as the reference plane relative to which the interface is positioned and the centre point of the interface. A CAD interface template is specific to the type of interface, connection it might specify and hence the technology used. Figure 134 provides an illustration showing two different CAD templates used for positioning and specifying both balls and rolls bearings.

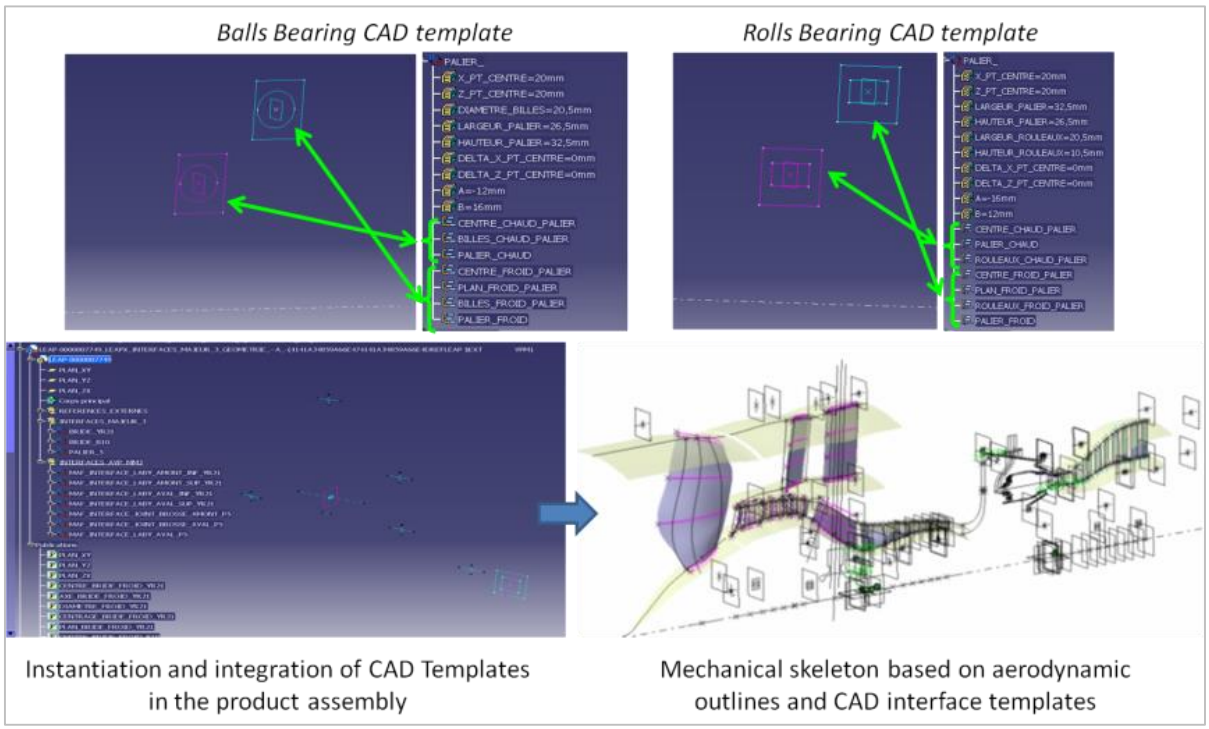


Figure 134: Balls and Rolls Bearing CAD templates

Afterwards the designers can start using the mechanical skeleton and design the product parts relying on the CAD interface template as shown on Figure 135.

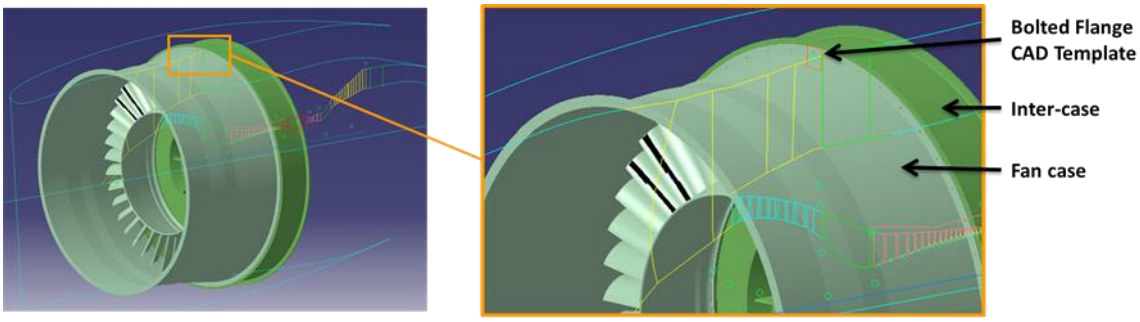


Figure 135: Fan case and inter-case designed in context using CAD interface templates

Since a CAD interface template is specific to a technology, we propose possible to link them with a standardized components catalog allowing, according to the specific design requirements and the dimensional constraints, to choose the appropriate technological component and to put it under the interface component in the tree structure.

Now, the designer might specify the topology of the interface to capture it for downstream applications. A CAD interface template, according to its technology, needs the expected mating features permitting to ensure this interaction. Indeed the template must also contain semantic information about these mating features. For instance, for a bolted joint with alignment, the user will have to specify the mating features ensuring:

- The planar contact that coincide with the bolted flange plan define previously by the template;
- The alignment face contacts (or interference if it is a clamped mounting);
- The co-axial constraint for the holes;
- And potential cylindrical and planar contact, clearance or interference with the bolt-nut assembly if already integrated.

These mating features (or mating faces) are captured directly on the CAD model topology of the interacting components through the use of **Publications** containing semantic information about the role of this mating feature. We propose to store the other interface features (function, application domain, type of connection) and other behavioural parameters in a **CAD connection template** that will also reference the mating feature and assembly constraint defined with the **CAD interface template**. Figure 136 shows an example of possible connection template form.

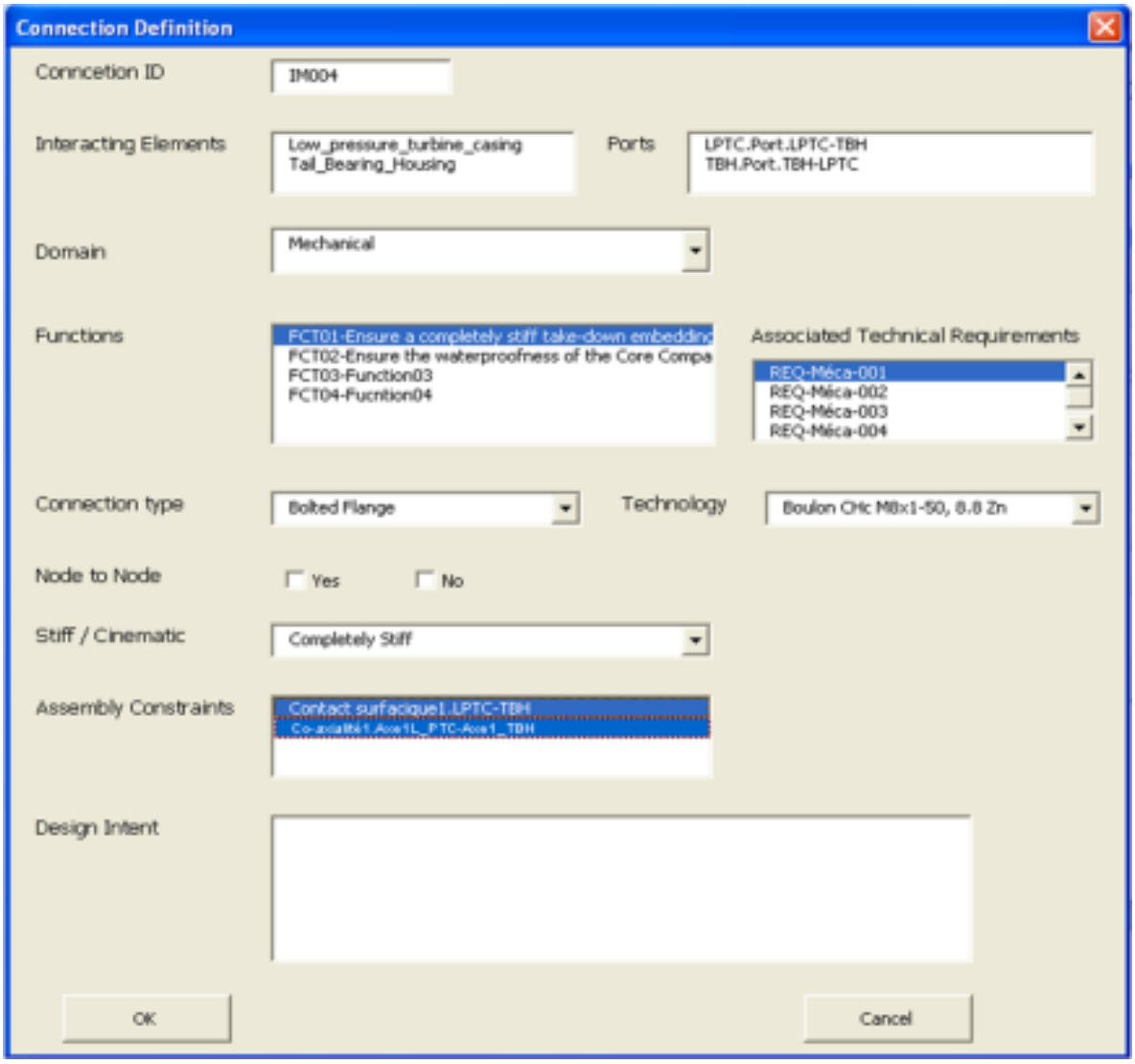


Figure 136: Example of connection template form

The synthesis of the proposed approach for scenario 1 is summarized and illustrated in Figure 137.

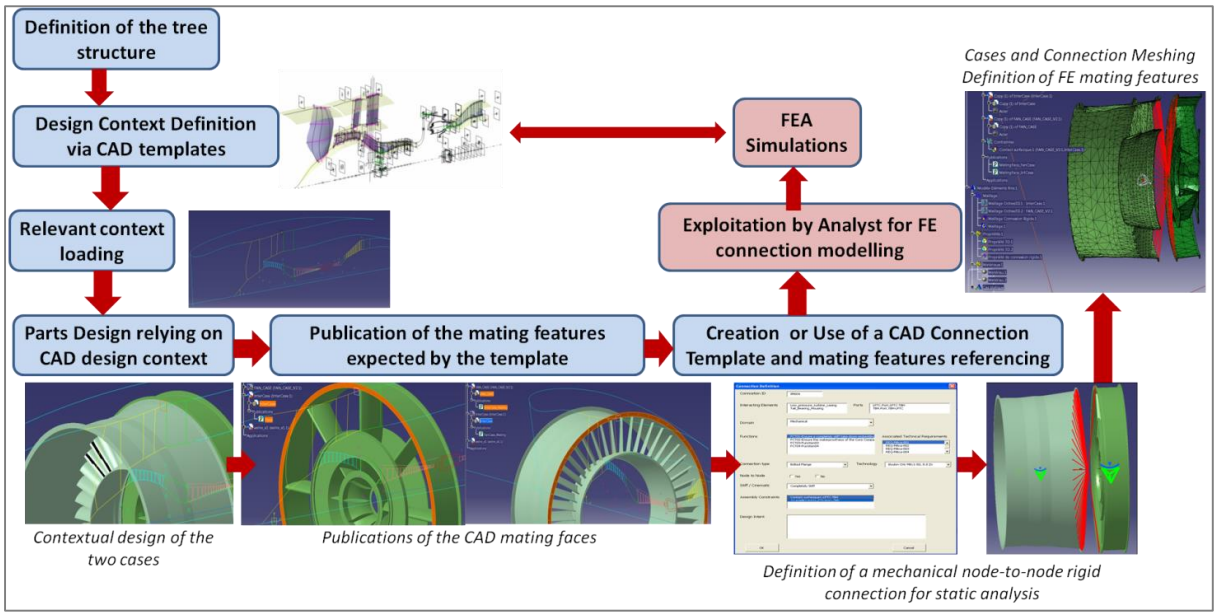


Figure 137: Proposed interface-based CAD design approach for enriching DMUs

We have identified another use case of CAD interface and connection templates that has to be investigated. The use case corresponds to the following scenario:

- The interface was not specified up-stream in the design process;
- The interface design intent results from a design change or from recommendations from a FEA analyses (e.g. stiffen the fan case with a bolted flange);
- The involved component's geometries are already defined.

The creation of this interface involves the sub-division of the component into two or more separated components linked by this interface.

A solution (illustrated in Figure 138) consists of automating the creation of this interface and the creation of the resulted new components based on specific CAD connection templates associated to operators (e.g. macro CATIA). The interface creation operator necessitates a certain number of topological features and parameters to help in positioning and dimensioning correctly the interface. The automation of the component sub-division(s) and the creation and integration of resulted components and interfaces within the product structure highly depend on the complexity of the interface and on the intelligence of the operator algorithm. For a bolted flange, when the interface template is correctly positioned and dimensioned, the operator performs the necessary geometry division following the bolted flange plan. Two shape bodies are hence created from the shape body of the cut part. Two new parts are as well created in the product structure. The shapes bodies are respectively transferred in the appropriate part models. And the cut part is deleted. If the operator can manage these operations, it can manage as well the publication of the mating features (at least the planar contacts which coincide with the bolted flange plan defined previously by the template). Such operators are conceivable but their development requires rich code programming resources and CAD modeller customisations to be applied on different type of interfaces.

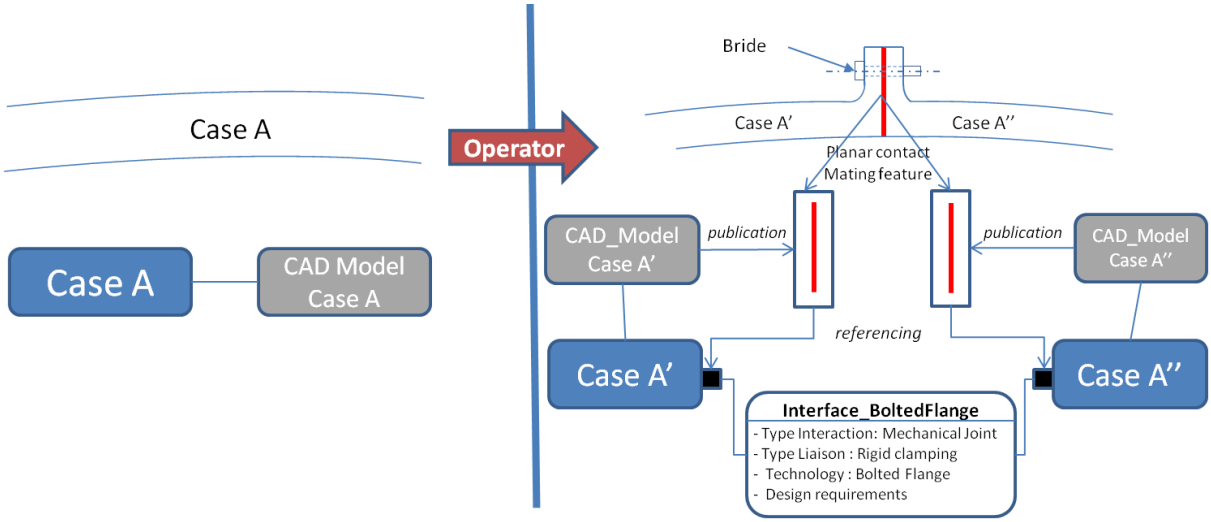


Figure 138: Suggestion for using CAD Connection Template for automatic creation of unexpected bolted flanges

2) Scenario 2: Automatic retro-fit interfaces capture from an existing and consistent DMU

In the second scenario RDMUs are already complete and coherent. However, we suppose that they have not been enriched with interface definition. In that case the manual capture of components mating features can be very tedious and time consuming on very large assemblies. Therefore, for this scenario, we recommend the use of automatic operators to capture these mating features. Three kind

of mating feature can be captured within a DMU: shape contacts, clearances and interferences and fluid solid boundaries (which are shape contact between a solid and a fluid domain envelope).

Several approaches can be whether found in the literature whether in existing commercial or open applications. An intuitive approach for the detection of geometric interfaces is to use Boolean operators between volume pairs of the DMU at hand. It has already been proven that this approach is inefficient on large assemblies and is not robust at all [Shahwan et al., 2013].

In CATIA the clash analysis tool performs the appropriate algorithm and is really efficient (even on large assemblies) but it is impossible to capture the topology of the mating feature. Nevertheless, as we will see in chapter 4, we have tried to exploit the results of the clash analysis to capture the mating features topologies calculating the bounding boxes of each solid “in clash” and identifying the faces that interact within the intersections of the bounding boxes of the two solids. The algorithm works but the operation on large assemblies is not robust and become inefficient. Limitations come from modelling constraints set by B-Rep modellers and that any face of a solid can be subdivided into smaller ones during a design process hence lengthening the procedure time. [Shahwan et al., 2013] propose to overcome this inconvenience by merging all adjacent surfaces and curves that share the same geometric properties (type, 3D location and intrinsic parameters). The authors have used the OpenCascade CAD (OCC) library to extract “conventional interfaces” (i.e. mating features).

Indeed, the OCC library provides an algorithm - “GEOMAlgo_Splitter”- for splitting a number of shapes with a set of surfaces [OCC, 2013]. In the python OCC community [PythonOCC, 2013] an efficient algorithm has been defined to exploit this function to extract a shape contact mating feature between two shapes (see Table 3). However, although it is an open source platform, this implies to be familiar to the OCC libraries and to python programming language in order to customise this function and test it on large and complex geometries. Even if we did not check this technology on large and complex DMUs, [Shahwan et al., 2013] and members of the community have underlined its limitations for providing the accurate mating prints.

```
def CylinderOnPlate():
  Box = BRepPrimAPI_MakeBox(gp_Pnt(0,0,0), 100,100,20).Shape()
  Cyl = BRepPrimAPI_MakeCylinder(gp_Ax2(gp_Pnt(50,50,20), gp_Dir(0,0,1)), 25,
50).Shape()
  Splitter = GEOMAlgo_Splitter()
  Splitter.AddShape(Box)
  Splitter.AddShape(Cyl)
  Splitter.Perform()
  result=Splitter.Shape()
  # find the shared face
  sharedface = None
  FSMap = TopTools IndexedDataMapOfShapeListOfShape()
  TopExp().MapShapesAndAncestors(result, TopAbs FACE, TopAbs SOLID, FSMap)
  Ex = TopExp_Explorer(result, TopAbs_FACE)
  while Ex.More():
    face = TopoDS().face(Ex.Current())
    index = FSMap.FindIndex(face)
    if (index!=0) and (FSMap.FindFromIndex(index).Extent()==2):
      sharedface = face; break
    Ex.Next()
  return result, sharedface
```

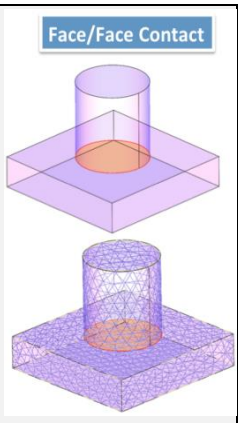


Table 3: GEOMAlgoSplitter algorithm used for extracting a mating feature between two shapes from [PythonOCC, 2013]

A certain number of commercial CAE applications (among which NX-CAE) offer the capability to capture these mating features pairing automatically coincident faces. However, since it is not related to the design intent and CAD interface features, the systems cannot guess what the mechanical behaviour is (glued, sliding contact, interference fit, etc.).

[Armstrong, 1994, Makem et al., 2012, Nolan et al., 2011, Nolan et al., 2013] use an opposite approach than the one conducted by [Shahwan et al., 2013] since they integrate in their CAD-FEA integration approach the concept of "cellular modeling" that consist in partitioning solid models into separate 'cells'. They use this decomposition for capturing what they call the simulation intent using these "cells" to apply specific boundary conditions, for automatic dimensional reduction and meshing of different kinds of topological areas (thin sheets, long slender regions, 'chunky' parts) and for extracting fluid volumes and fluid-solid interfaces. Their extraction algorithm permits to capture an accurate print of the mating features (not only the two faces in interaction but their intersection). Illustration of their approach is given in Figure 139 below.

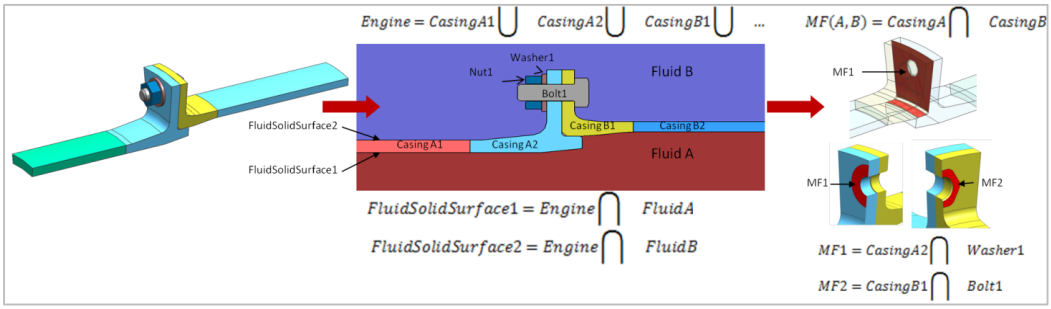


Figure 139: Cellular Modelling concept enabling CAD mating features extraction adapted from [Robinson et al., 2011]

Concerning the extraction of fluid domains their approach consist in using fluid domains for designing space rather than being derived from solid parts or gas paths. However, their approach has not been tested on large and complex assemblies such as an aero-engine DMU with a multitude of equipments crossing the fluid domains. Other CAE applications such as the ANSYS DesignModeler have fluid extraction tools as illustrated in Figure 140 below.

Figure 140: Fluid domain extraction demonstration from [ANSYS, 2013]

10.3.3.4 Use of the DMU interfaces in the preparation of FE model

The aggregation of all the information contained in both CAD interface and connection templates constitute the design intent of the interface on which the analyst can rely to model FE connections. The objective of this proposal is as well to integrate in this definition features supporting the definition of system behaviours:

- The interface behavioural modelling parameters (e.g. interface mesh specifications expressed through the use of CAE mating features),
- The finite element (FE) representation of the interface specific to the analysis domain that are derived from the interface design properties (function, technology, design intent) - e.g. a rigid beam node-to-node connection for a bolted flange, combination of a spring and rigid body element for a bearing, etc.,
- The behavioural parameters of the connection (e.g. the stiffness/rigidity of a bolted flange),
- The behaviour at the interface (degrees of freedom, boundary conditions, etc.).

The different FE connections are made by using the different interfaces which are specifically involved for the discipline studied; for example, in mechanical simulation, and following the system level of interest, only specific interfaces are used so as to represent the mechanical behaviour of the product. In an attempt to use interface for FEA simulations, a specific extraction of the information from the DMU has to be done to fit for the simulation. This information has to be exploited for the preparation of the simulation model, from the DMU to the Finite Element modeller. The exploitation of the DMU interfaces for FEA is performed in two steps:

- Localisation of the interfaces to be used in the finite element modeller: the interface might reference the components in interaction and associated CAD mating features publications which indicate the portion of the component geometry the interface is connecting.
- Translations of the design intent into functional and design parameters linked to the simulation discipline and objectives.

This precise and meaningful information can be used for downstream processes, like the aggregation of the individual meshes (detailed later on). The goal is to be able to set up automatically the finite element model of the interface that ensures the connection between two components. Our proposition is to be able to use any information contained in the DMU and translate it into a behaviour definition of the connection. Therefore, we need to define some generic objects in the data model that can be suitable for both design and analysis definitions.

Then FE mating features can be defined in two following ways:

- From CAD model referring to a geometric element that has been published. The information is published on the CAD model is propagated to the mesh of the finite element model. Created on the mesh, the FE mating feature publication will reference groups of mating finite element nodes that will be used to connect the model with the interface. Based on the CAD interface topology and the design intent, automated decisions can be made about the nature of the MPCs[2] based on the type of interface used (Cylindrical to cylindrical shear connection and Planar to planar axial connection) see related works from [Nolan et al., 2013] and illustration in Figure 141 below.

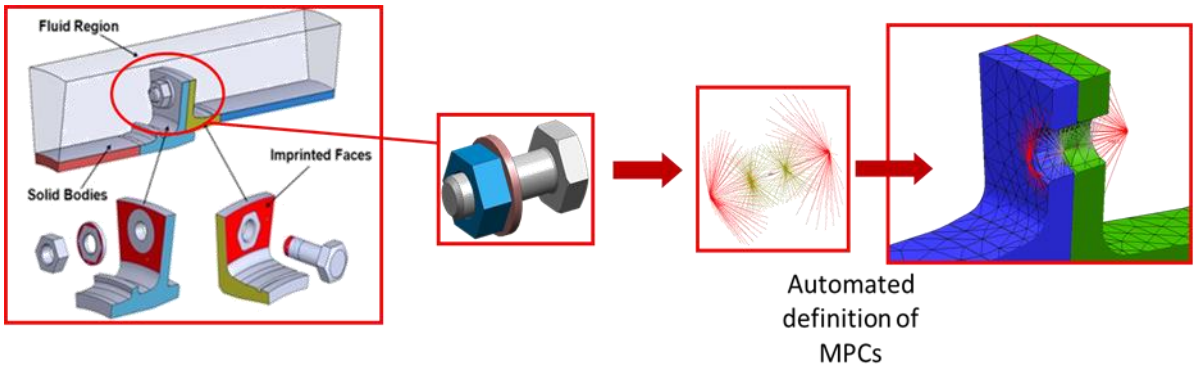


Figure 141: Automatic bolted flange FE modelling based on design intent [Nolan et al., 2013]

[2] A MPC (multi-point constraint) is a constraint that defines the response of one or more nodal degrees-of-freedom (called dependent degrees-of-freedom) to be a function of the response of one or more nodal degrees-of-freedom (called independent degrees-of-freedom). MPCs specify the possible displacement of a node with respect to the displacement of other adjacent and related nodes. MPCs can be used to model certain physical phenomena that cannot be easily modelled using finite elements, such as rigid links, joints (revolute, universal, etc.), and sliders, to name a few. MPCs can also be used to allow load transfer between incompatible meshes. However, it is not always easy to determine the explicit MPC equation that correctly represents the phenomena you are trying to model.

- Directly from a mesh, having the ability to define nodes, finite element, finite element faces that will be referenced by a **FE mating feature publication** and potentially associated to the corresponding **CAD mating feature publication** if available.

The Figure 142 below illustrates how the CAD mating features and related design intent (specification of the interaction taking place at these mating features) are used in a FE modeller to define the FE connections.

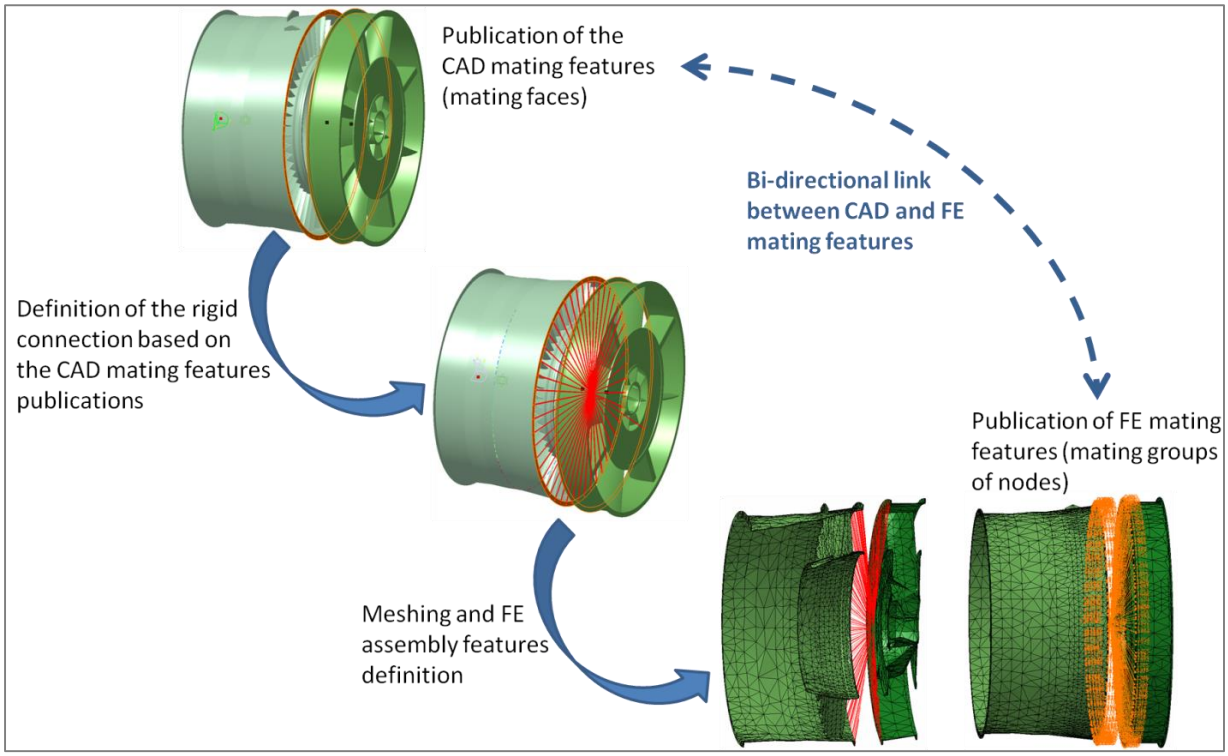


Figure 142: Use of CAD mating features and related interface design intent to define FE models connections

Another advantage of enriching DMUs with interfaces definitions is that can serve as a basis for re-structuring more or less automatically a product structure. Indeed, we believe that semantically enriched directed graphs specifying the interactions between components can help in automating this required structure reorganisation according to captured business specific structuring rules. However we propose to use the same kind of approach using an object-oriented system modelling language.

10.3.4 Digital MBSE system modelling and integration framework

10.3.4.1 Framework for defining system architectures and system models

The system modelling framework that we propose to develop is focused on the definition of logical, physical and behavioural system architectures and especially **CAD/DMU and CAE/BMU model architectures.**

Our approach consists in developing a system modelling framework within PDM systems and coupling it with the use of DMU and BMU representations and to assist designer/integrators in:

- Organising and configuring multi-level and multi-domain system architectures;
- Capturing and exploiting both design and simulation intents of system artefacts (and in particular system interfaces) by accessing and capturing features present in CAD and CAE models and referencing them in the PDM system;

- Supporting the functional, structural, topological and behavioural specification of system interactions and interfaces including multiple levels of hierarchy via ports and ports delegations optimizing the integration/assembly activities;
- Integrating or referencing more or less automatically and consistently CAD design data sets with the associated CAE data sets in order to ensure a traceable design/simulation information chain.

The objective is to enable the exploitation of the resulted system architecture models for:

- Building and/or loading the corresponding DMU representation: using CAD models assigned to each system blocks, positioning attributes assigned to each hierarchical link and assembly constraint define on interfaces;
- Building the corresponding finite element BMU representation: using FE models, using interface definitions for automating DMU structural transformation and using a complete behavioural system architecture model to automate the assembly of FE models;
- Maintaining consistency between multi-level and multi-domain product views: relating system blocks via hierarchical, interaction or dependency links.

As shown in Figure 143, the proposed modelling formalism is similar to the internal block diagrams as defined in SysML language. From a user's perspective, the resulted system model is a representation of the product architecture describing the various product components and their relationships. These components are represented by "**systems blocks**" or "**interface blocks**" interconnected by **connections** via **ports**. Each of these system objects allow to access specific engineering data sets.

The **system blocks** are themselves representations of a system. In other words, this representation is multi-level: a system consists of sub-system and so on. One system blocks represents a usage occurrence of system artefact design definition with its corresponding analysis definition. It hence enables to access to the appropriate design and analysis definitions data sets of the system artefact.

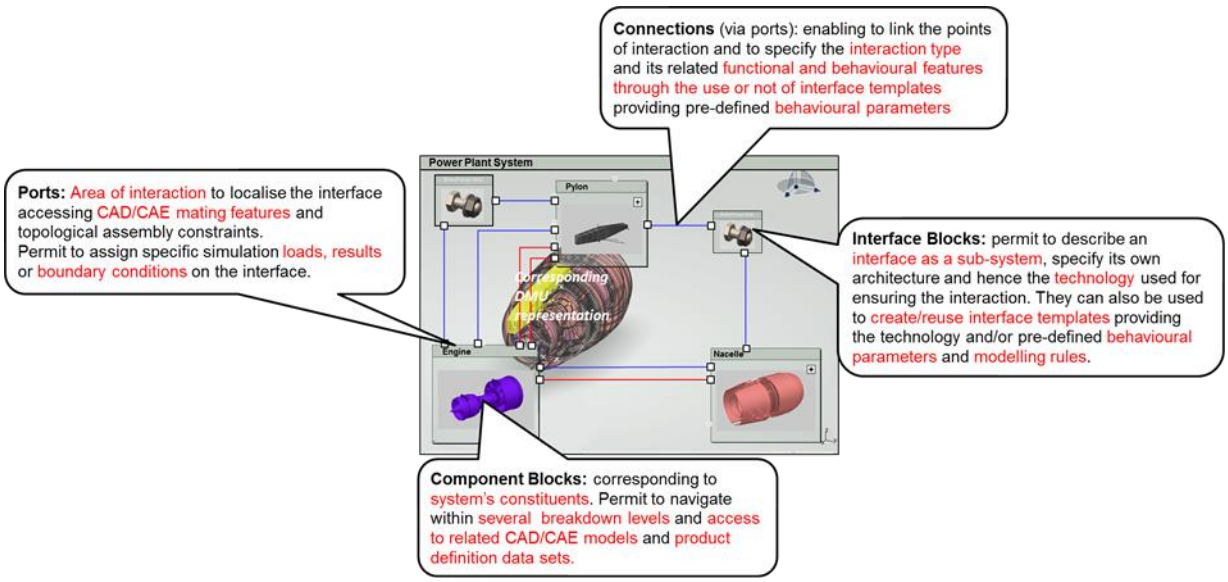


Figure 143: Proposed MBSE framework formalism and related objects

The **connections** or **connectors** define the physical interactions between components and capture the design interaction properties and hence the related behaviour features of the interface (linked to the technology used). These features permit to specify the behavioural representation (i.e. the way to

model the FE connection in pre-post tools). When a technological or physical component/system is required to ensure an interaction (i.e. a connection), the system model can be enriched with "**Interface Blocks**" which behave like another component block since it is system composed of different sub-systems (e.g. a bolted flange composed of a ring of bolt-nut assemblies themselves composed of a bolt, a nut and a washer). These interface blocks permit to access to the functional and technological description of the interface and allow multi-level representation of the interface system. They can be captured in and created from a stored library of interface models.

The connection between components is modelled using **ports**. Ports are elements of interaction of a component connecting it to another component through a component interface. We propose to use port objects to specify the interaction area; i.e. the geometric or topological definition of the interface. They hence consist of a geometric specification using features extracted from CAD models (e.g. mating faces) as well as for interface mesh specification using CAE mating features. CAD and CAE mating features associated to the same port should represent the same topology and hence will be associated. Additional information such as modelling requirements and simulation results can be associated to these port objects to help integrators to maintain consistency between models to assemble or between interdependent simulations.

As illustrated in Figure 144, two main types of ports connections (or connectors) have been defined:

- The physical interaction: which permit to encapsulate the design interaction properties and hence the related behaviour features of the interface.
- The ports delegation: Interactions at outer ports of a parent block are delegated to ports of children blocks. To define a port delegation, ports must match (same kind, type, direction, etc.). Connectors can cross a component block boundary without requiring ports at each level of nested hierarchy. Port delegation can be used to preserve encapsulation of block (black box vs. white box).

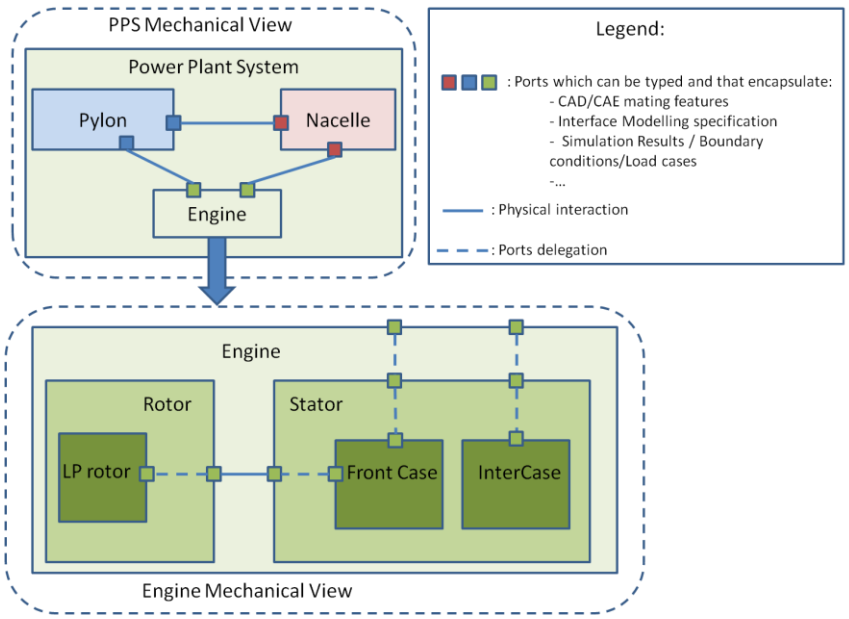


Figure 144: Physical interaction and port delegation within a system model internal block diagram

This system modelling framework enables to breakdown the studied system and sub-systems into multi-level breakdowns. According to the pre-established partnership work-sharing rules, system or

sub-systems integrators can share/exchange a more or less detailed view of the sub-systems in interaction.

Figure 145 shows an IPPS DMU mechanical system view composed of a pylon, a nacelle, an aero-engine and the various mechanical interactions between these sub-systems. For instance, and as illustrated on this view, the IPPS integrator does not have access to the detailed definition of the aero-engine. The engine block is exchanged / shared as a “black box” only exhibiting the interfaces and interactions with the pylon and the nacelle.

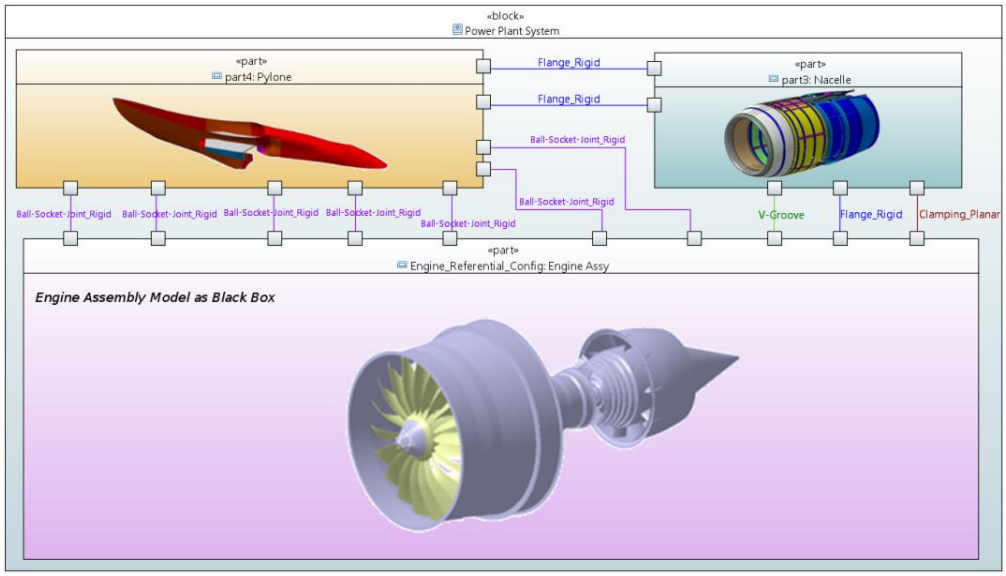


Figure 145: Example 1 of an IPPS DMU mechanical system integrator view - Engine block shared as black box

Figure 146 shows another DMU system model of the same system. In that case the system integrator needs to have access to a more detailed view of the engine in order to know to which engine modules the nacelle and the pylon interfaces will be attached. However the integrator does not have access to the detailed definition of these modules.

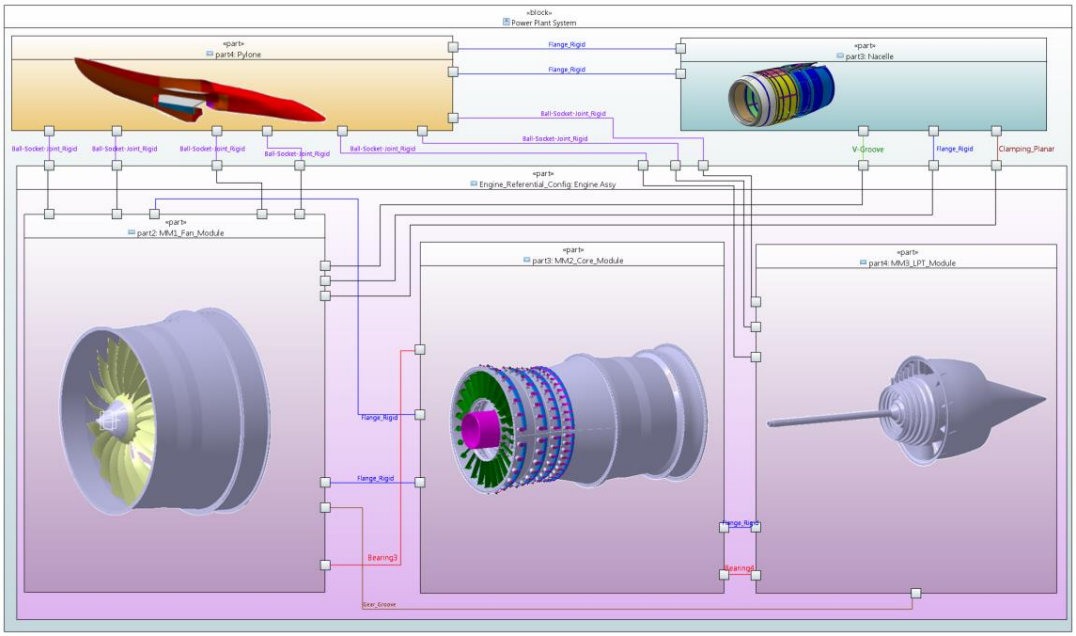


Figure 146: Example 2 of an IPPS DMU mechanical system view - Engine modules blocks shared as black boxes

In the example of Figure 147, the integrator now has access to the detailed definition of the engine system block. However, the engine block is not yet configured for mechanical analysis in this view. The mechanical system integrator will need to perform some structure reorganisations to use this system view for the future mechanical analysis (see Figure 148).

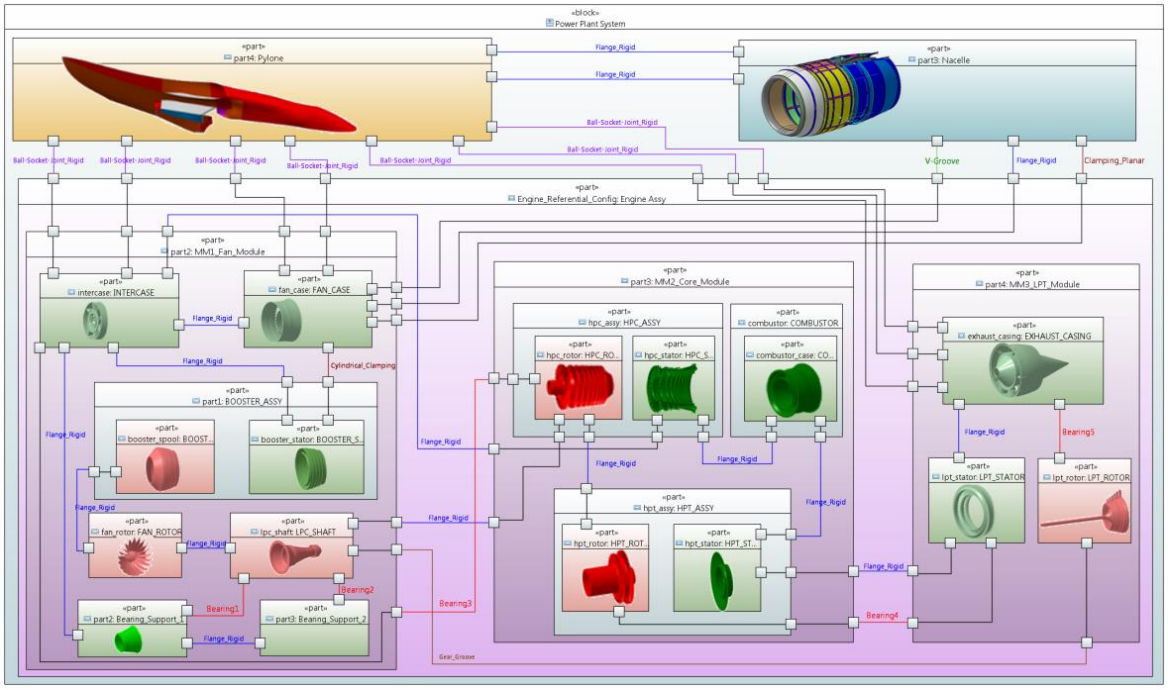


Figure 147: Example 3 of an IPPS DMU mechanical system view - Engine block shared as white box in initial configuration

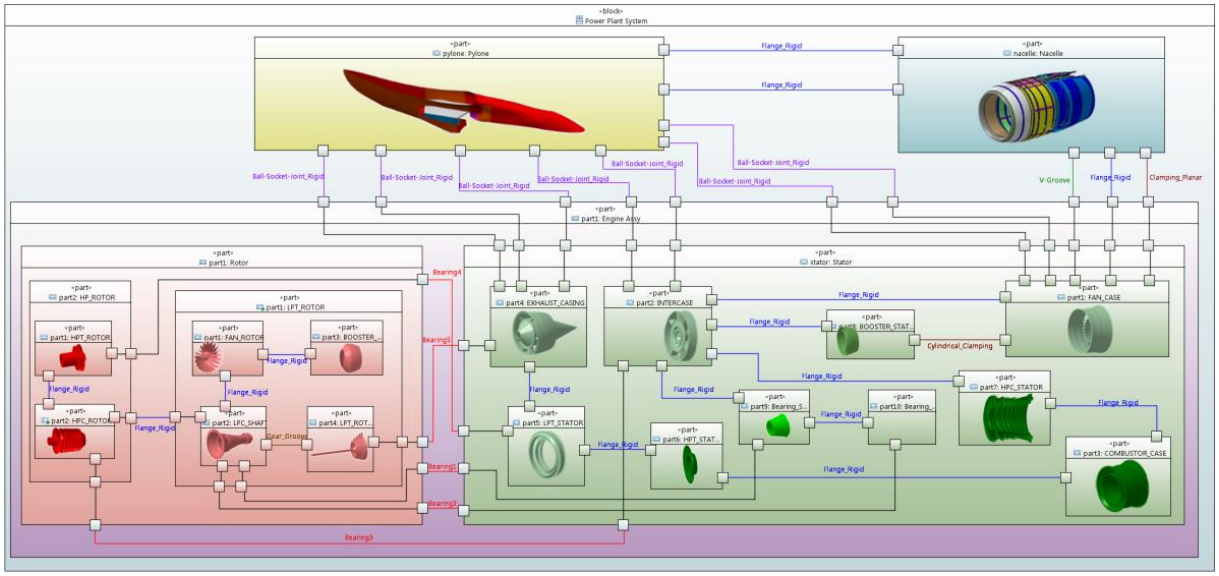


Figure 148: Example 3 of an IPPS DMU mechanical system view - Engine block shared as white box in mechanical configuration

10.3.4.2 Framework for specifying components interfaces and interactions

We propose to couple the interface information model proposed by [Sellgren, 2006a] with the use of an object-oriented system modelling language such as SysML. In SysML, the system architecture is made of **building blocks** connected by **connections**. Each of these connections relates two **ports** of the connected components. In SysML, these ports represent energy, data, material or signal flow. We propose to use this visual modelling language but extend its usage using ports and connectors to specify

the functional, geometric/topological, technological and behavioural system interfaces. The idea is to provide a better support to define hierarchical links (multi-level system blocks) and the physical interactions and/or assembly constraints/features between components but above all between their **design and simulation models**.

Figure 149 illustrates the concept giving the equivalent meaning of an aero-engine's fan case 3D assembly with the proposed system modelling language. This example also illustrates the links between the system architecture objects and related numerical engineering data.

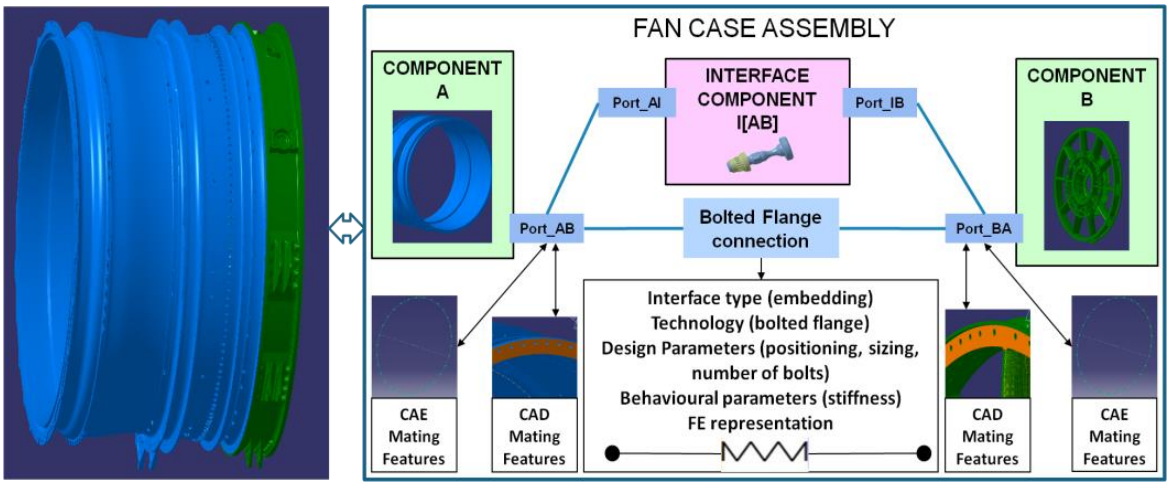


Figure 149: Links between SE objects and engineering data – Example with a FAN Case assembly

Figure 150 provides another illustration of the above example. On the left side of the picture, it represents the physical view (CAD and CAE) of the assembly. In the middle is the equivalent meaning of this assembly in the system view. And on the right side are the corresponding used engineering data which need to be associated to the different system framework's object in order to specify the interface. The result is the integrated FE model presented in the physical view (the red one), which should be automatically generated from the definition given in the system framework.

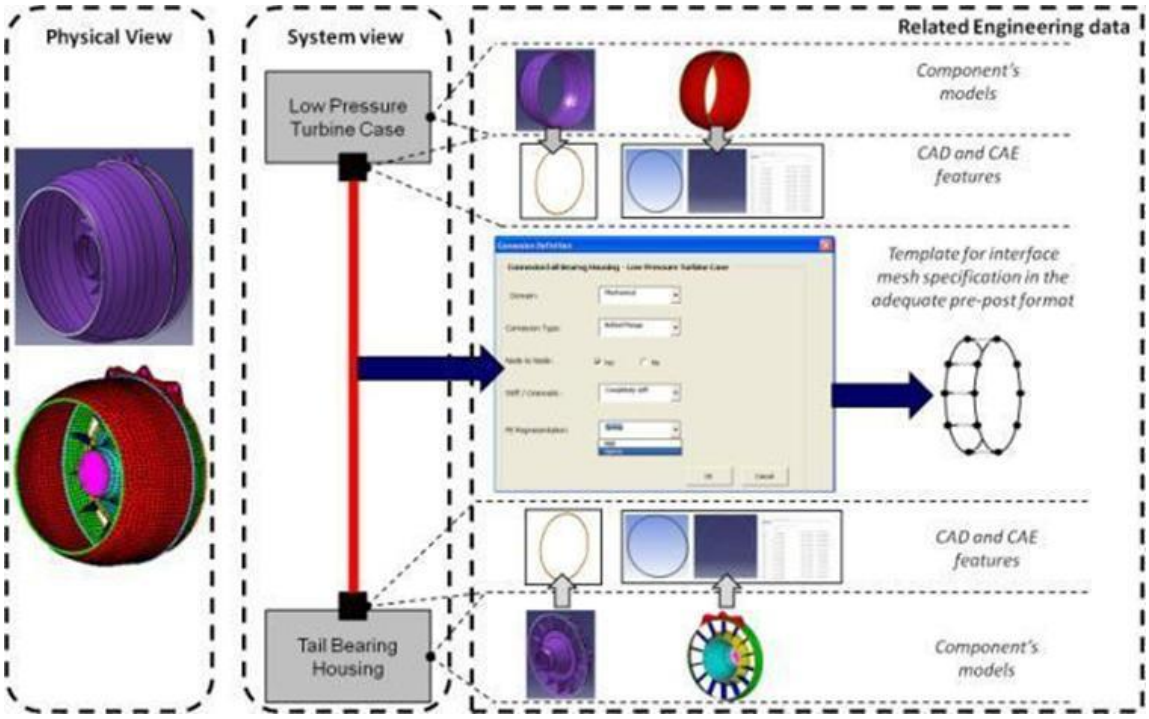


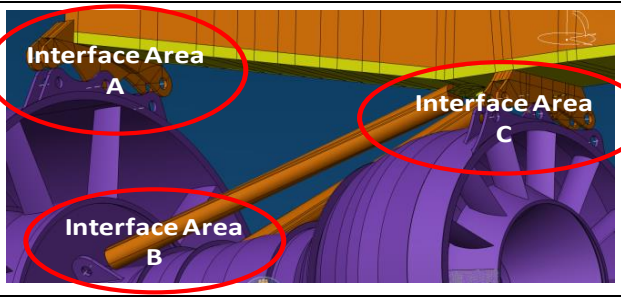
Figure 150: LPTC-TBH assembly specified with the MBSE integration framework

Table 4 synthesises the content of the different system framework's objects enabling to define a model-based product/system representation encompassing the definition of the interactions between two system's constituents.

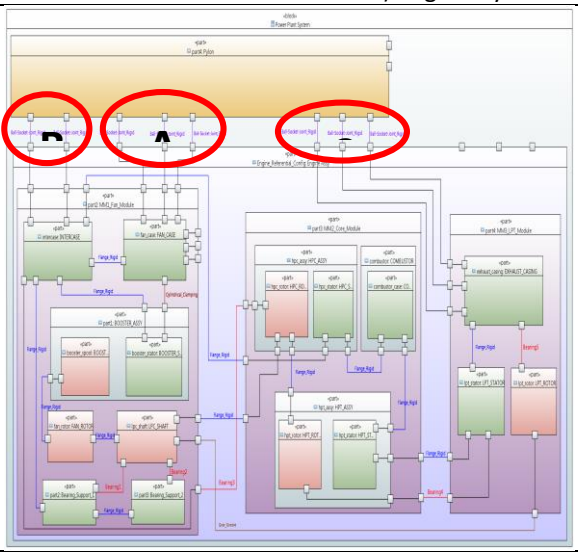
System Model Object	Associated engineering data
Component Block Structural, geometrical and physical definition of system components	Component's CAD and CAE models
	Associated physical parameters
Interface Block: Functional and structural specification of the interface	Function of the interface
	Technological definition and design intent
	CAD model of the interface component (technology)
	Interface's sub-systems blocks if the interface is a system
Port: Localisation and geometrical specification of the interface	Assembly Geometrical Features
	Mesh specifications: mesh type, mesh size, imposed meshed surface, nodes group and file describing the nodes references, numbering and positions.
	Boundary conditions and load cases
	Interface Results: which need to be cascaded and used as boundary conditions for downstream simulations.
	Interface design requirements
Connector: Behavioural Interface specifications	Domain definition: defining the type of connection (Mechanical, Thermo-mechanical, Fluid-Fluid)
	Associated behavioural parameters
	Generic FE link representation (in neutral format)
	FE link model (in authoring format)

Table 4: Description of the system framework's objects content - Links between system model entities and engineering data

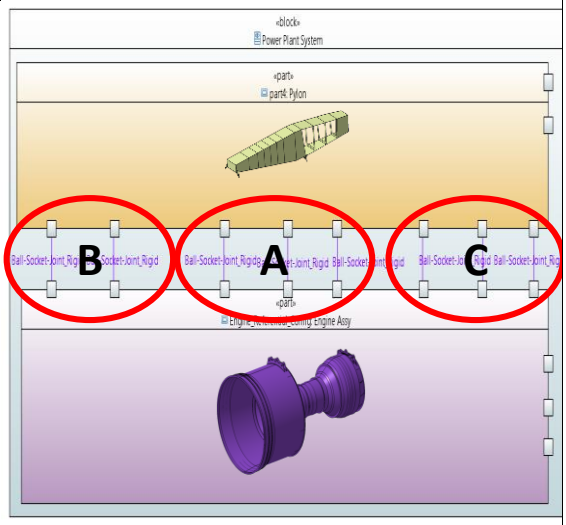
The benefit of using interface system blocks is to be able to describe an interface as a system. To illustrate this concept, Table 5 shows various system representation of an engine-pylon DMU assembly model (view A). In the view B, the engine block is shared as a white box, whereas in the view C it is shared as a black box. The view D includes the mount system blocks ensuring the interaction between the engine and the pylon. In that case, the mount system blocks are shared as black boxes. The view E includes a detailed definition of the mount system and its compoents interaction.



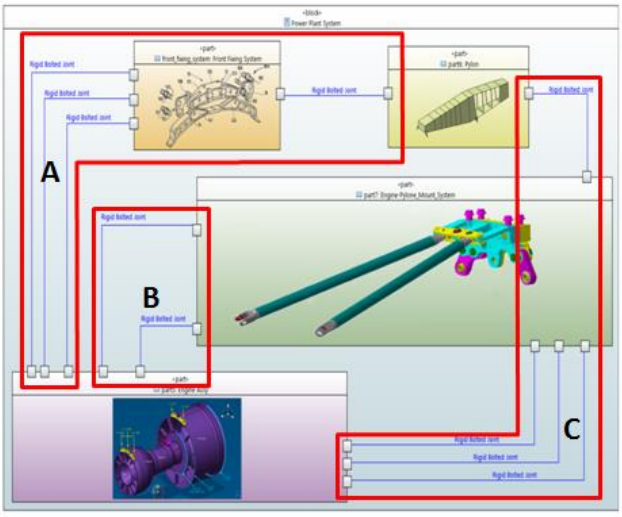
A) Engine-Pylon DMU assembly model



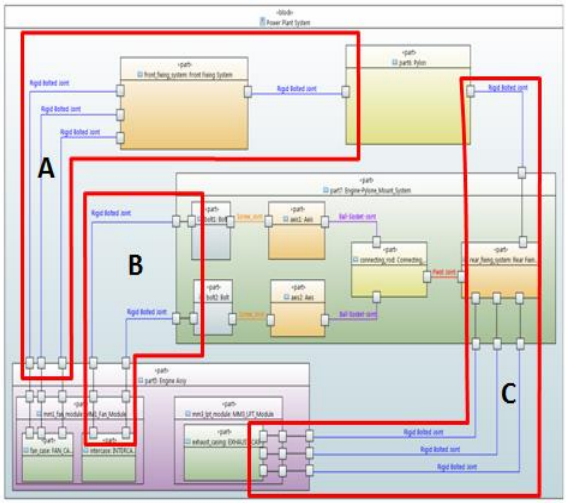
B) Engine-Pylon mechanical assembly system view with the engine block as white box



C) Engine-Pylon mechanical assembly system view with the engine block as black box



D) Engine-Pylon mechanical assembly system view inluding the mount system interface blocks shared as black box.



E) Engine-Pylon mechanical assembly system view inluding the mount interface system blocks shared as white box.

Table 5: Engine-Pylon assembly model described with different system views – illustration of the use of interface system blocks for defining the mount system

For the preparation of the simulation model, interfaces have to be automatically derived from the information kept in the DMU, and exploited in the finite element modeller. The extraction can be considered as an automatic translation of the design intent of the interface in a simulation model that describes the function performed by the interface. The goal is to be able to set up automatically the finite element model of the interface that ensures the connection between two components.

Our proposition is to be able to use the information contained in the DMU and to translate it into a behaviour definition of the connection. Therefore, we need to define some generic objects in the data model that can be suitable for both technical definition and simulation. Generic objects are defined in the system representation of the product with the object "Port" and "connectors". **"Ports" and "connectors" are elements that materialize the definition of the interface and ensure the association between the technical definition and the behavioural definition**.

The "port" enables to define and locate the area of interaction (represented by features) of the component's representation (CAD or CAE model). It is used to identify where the connection takes place. To ensure the capture of this information, the proposed solution is the "**publication**". The publication is a reference that points out an element or a group of elements contained in a CAD file or in a Finite element model. These elements can be surfaces, vertex, lines, curves for CAD mating features and nodes and elements for CAE mating features. The publications capture the information about the location of the position of the CAD or FE elements/features used to connect the interface. These publications (for CAD model and for FE model) are attached to the object "port". Ports ensure the association between the definition of the location of the interface in CAD model and its representation in FE model. This information needs to be extracted and specified to partners in order to ensure that they model appropriate and well located connection.

For the simulation, the design intent of the interface, which is composed by the function of the interface and technology used to manage this function, is translated in specific information:

- Function and Role:
  - Simulation type (Mechanical, Thermal, etc.);
  - Nature of the connection (rigid embedded connection, thermal convection, etc.).
- The technology used (Bolted-Flange, bearing...):
  - The finite element model of the interface;
  - The parameters describing the nature of the connection and its modelling requirements (node-to-node connection, MPC type, etc.).

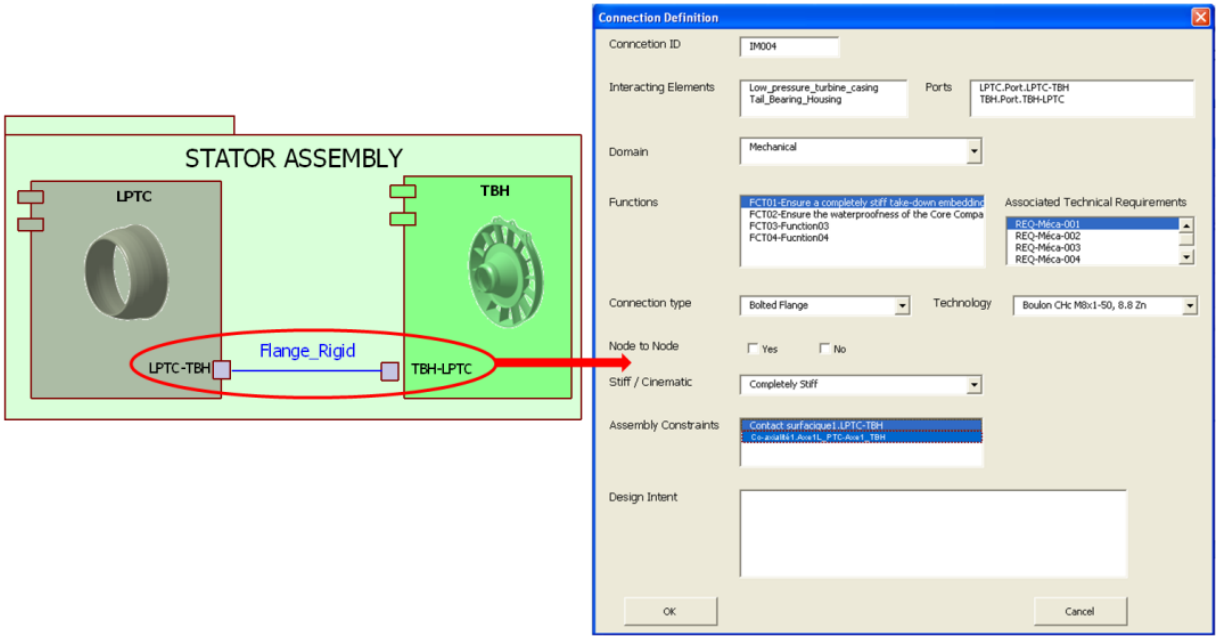


Figure 151: Interface design intent definition example

Once the simulation has been performed, the results are extracted from the interface. Interfaces have a particular role in the simulation of system, because they describe how the system behaves. In

system engineering, the results are derived from the interfaces and transposes to different component connected to it. If we consider a system which is composed of sub-systems linked together by interfaces, the interfaces must respect some specifications regarding the system. But for the sub-system the result of the interface is used as a boundary condition of the sub-systems simulation.

For large and complex systems, component and interface definitions are generally multi-instantiated or used in different system architecture corresponding to specific product business view points. It is therefore of primary importance to provide a library of interfaces in order to be able to re-use parameterized interface templates encapsulating the physical and behavioural modelling parameters of an interface. The idea is to support the analyst to use appropriate interface system model according to its simulation intent.

10.3.4.3 Automatic FE assembly modelling based on system models of behavioural architectures

The translated design intent from DMU described previously contains the data and information needed to create the simulation model. These data are associated to the system view dedicated objects (blocks, ports and connection). All these information have to be used in the different simulation authoring tools (Pre processor and solvers).

Once the CAE models have been provided and validated through an appropriate quality assessment process, the analyst need to check if all requested models are gathered and assigned to the appropriate elements of the analysis system view (Block, ports, and connectors). Now, the specified interfaces design intents need to be translated into simulation intents; that is to say into modelling hypotheses and specifications according to the nature and objectives of the simulation to perform.

These behavioural descriptions of interaction between components are performed directly through the system view using an interface library. Dedicated parameters and finite element models are assigned to system model objects. This information, ensure the description of the behaviour regarding the type of analysis to perform. For instance, the behavioural descriptions of a bolted-flange are defined through a linear finite element with stiffness parameters that connect two features of the component's CAE models.

When the information required to assemble the simulation model is defined, some routines are needed to create automatically from the system view the related translation to a pre-processing tools (as illustrated in Figure 152), in order to compute it in the solver.

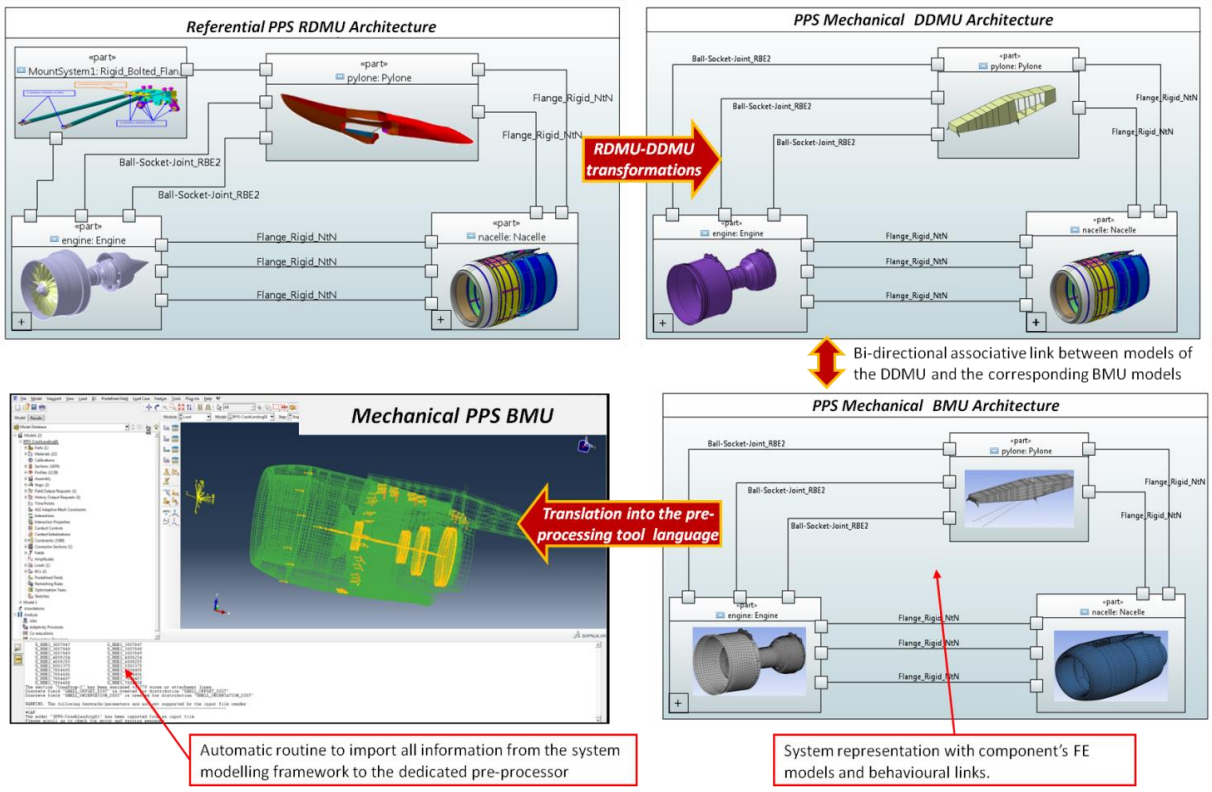


Figure 152: Automatic assembly of FE models based on a BMU specified in the system modelling framework

10.3.4.4 Management of multi-disciplinary product views through the use of a system model

Section 10.3.4.1 shows how the MBSE integrator modelling framework enables defining, managing and linking different DMU and BMU multi-level system view points. Next step was to investigate how this framework could be used not only to manage and link multi-level view points but also multi-domain and multi-disciplinary view points.

Table 6 gives an illustration of two different viewpoints for creating and structuring an integrated thermo-mechanical FE model for the power plant system. On these two view points, the thermal FE models or results are coupled with the structure of the integrated mechanical model. This is done to take into account fluid/structure interactions and hence heat flow impacts to make the structural material properties of the PPS structure more realistic. In the first viewpoint, the thermo-mechanical FE model consists in mapping temperature fields as boundary conditions on specific area of each of PPS subs-systems (the engine, the pylon and the nacelle) mechanical FE models. In the second view point, we first create the two integrated FE models (mechanical and thermal models) of the PPS. Then a mapping of temperature fields is performed directly from the integrated thermal model results to the integrated mechanical model. These two viewpoints highlight the fact that even for doing the same type of simulation on the same system there exist several ways of doing it according to the architect model or model integrator viewpoints and constraints. For instance, the second architecture model should theoretically provide more accurate results and be more efficient (less mapping to perform) than the first architecture model. However, in the context of a collaborative and distributed process, there exist many more constraints to integrate heterogeneous thermal models in different data formats. The first model architecture is easier to handle because it just requires defining temperature fields (resulted from the thermal analysis of each sub-systems and not from a bigger integrated model) as boundary conditions of the integrated mechanical model.

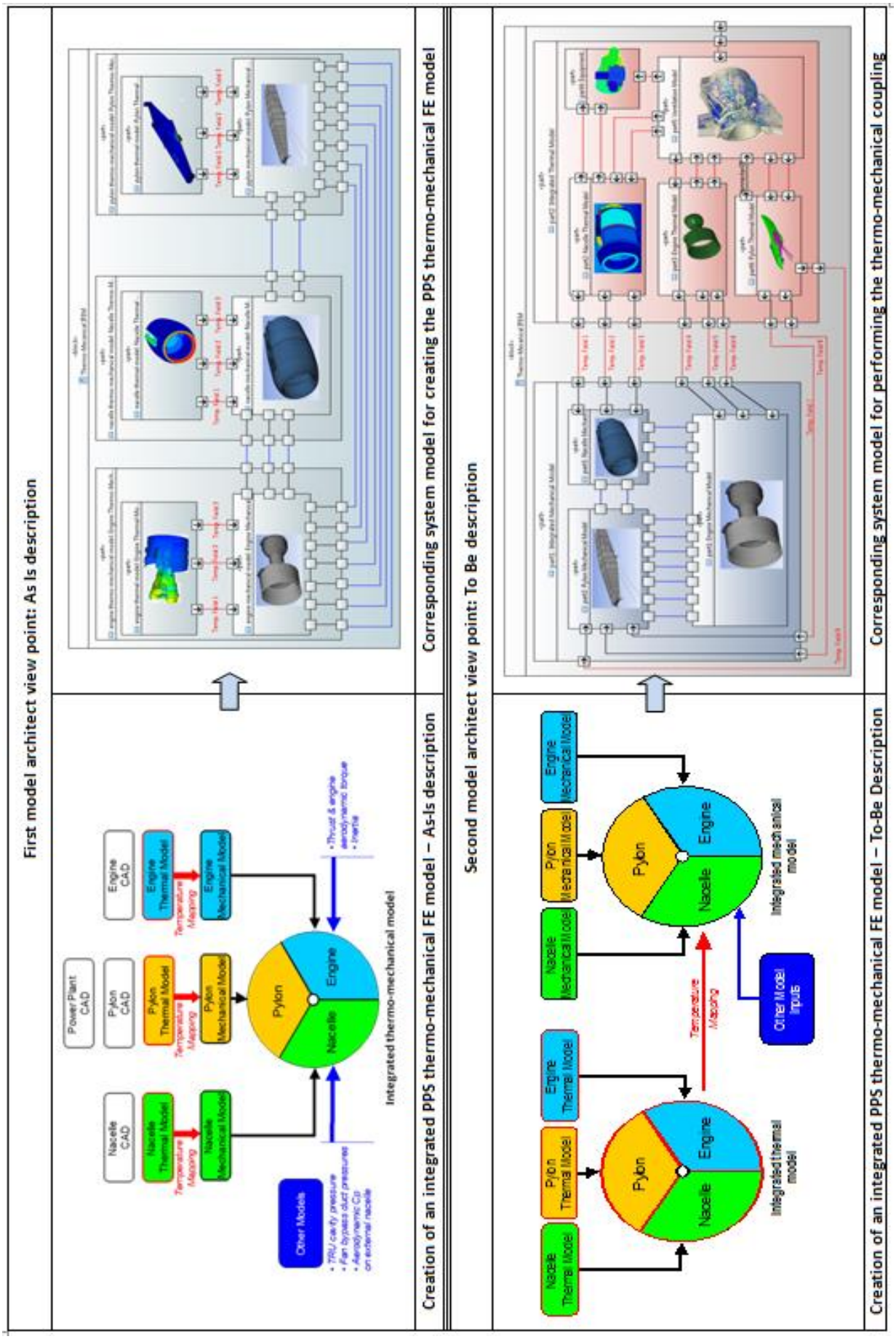


Table 6: Creation of an integrated thermo-mechanical model for the power plant – two different model architect view points

10.3.5 CAD-CAE data integration within the DASIF framework

10.3.5.1 Definition of the simulation intent

The first step of generic CAE process is the definition of the simulation intent. Defining a simulation intent consists in several steps that the analyst performs in order to efficiently run its analysis process and enable first the reuse of appropriate CAD and/or CAE artefacts and secondly the automation of some modelling procedures (as illustrated in Figure 34). Figure 153 and Figure 154 respectively illustrate the process of defining simulation intents and the simulation intent data package that we propose to implement. The process aims to be generic, but the M&S (Modelling and Simulation) hypotheses and specifications package is specific to the type of analysis and models used (in our case FE models).

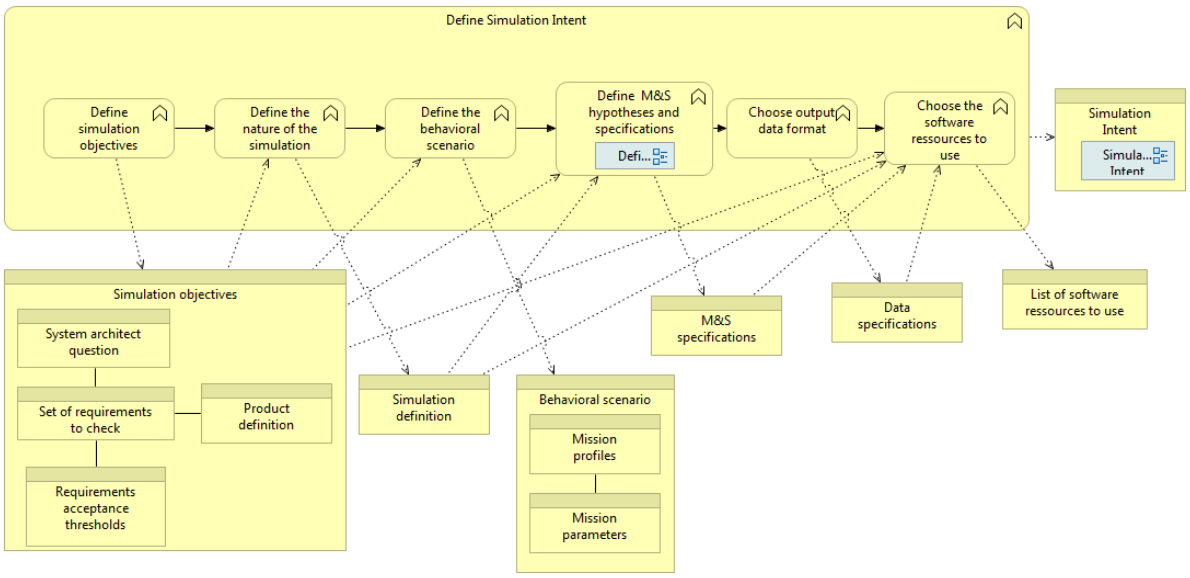


Figure 153: Generic process for defining a simulation intent

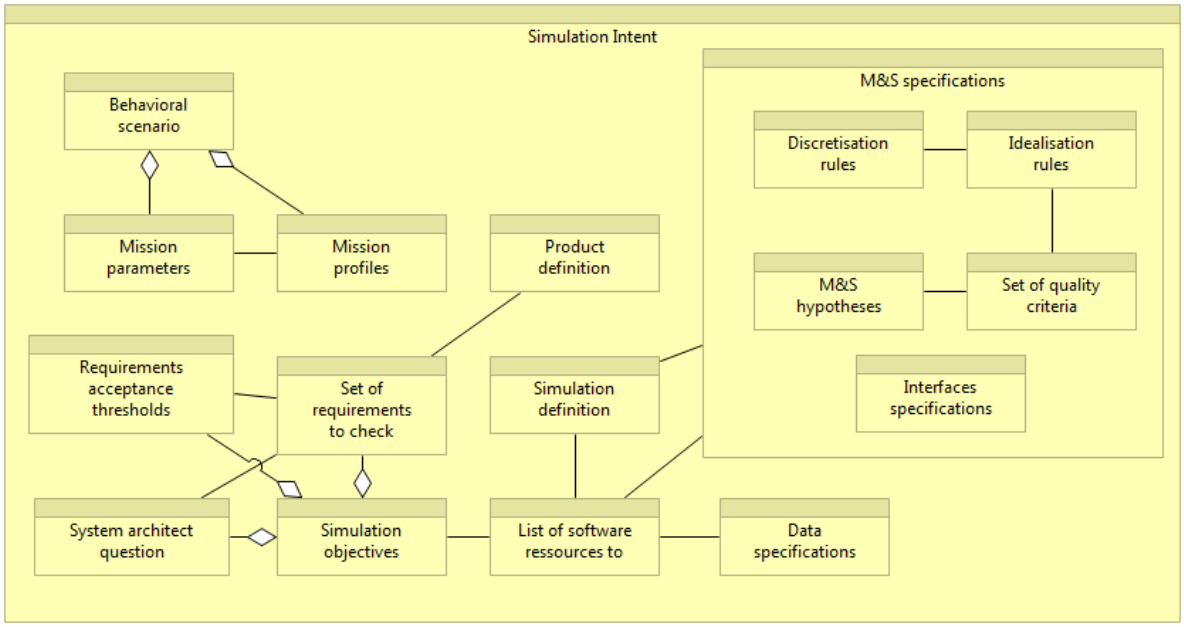


Figure 154: Simulation intent data package for FEA

10.3.5.2 Integration of CAD-CAE data through the use of the MBSE modelling framework

In their own environment, the various simulation integrators and designers can use the MBSE modelling framework as simulation definition referential, containing the whole description of their simulation models architectures. In the first step of the simulation process, the simulation components identified in the above architecture aim at representing the issue to be analysed in a given discipline view (for instance in the form of a mesh-like model) and resulting from a CAD data transformation process. So there is a link between the simulation components (models) and the DMU components (CAD parts), that need to be traced. That is why this framework must be perfectly synchronized with the corresponding Digital Mock-Up and product definition. But, as mentioned above, the added value for integration activities, is the capability to manage all the behavioural configurations corresponding to fit-for-purpose behavioural mock-up for a specific simulation, as well as to integrate in a single system referential CAD and CAE data, ensuring traceability and enabling easy re-use of models.

10.3.5.3 Definition of the associative CAD-CAE model network

The AMN (associative model network) concept is derived from works performed in the frame of the CRESCENDO project (see Figure 155).

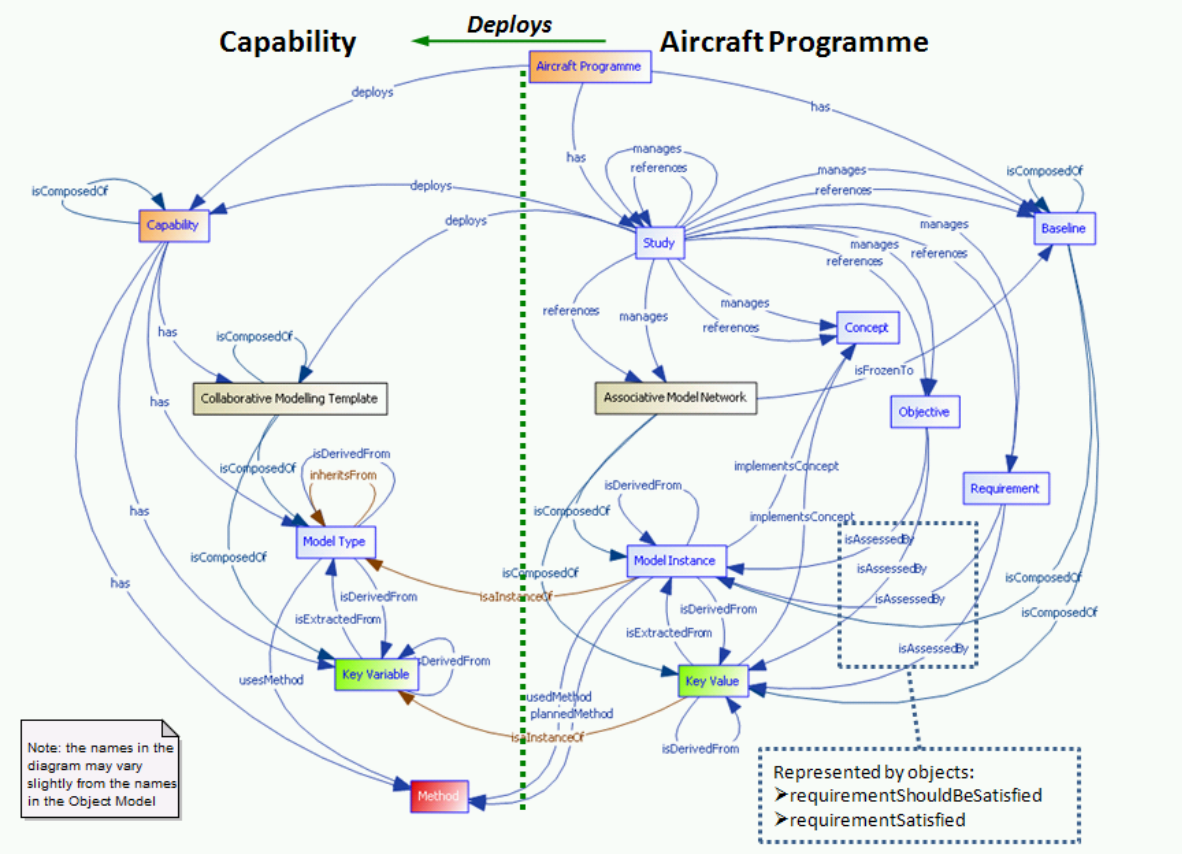


Figure 155: the AMN concept as defined in the BDA Business Object Model from [CRESCENDO, 2013]

An implicit concept appears in this figure: the high level definition of the simulation intent defined here by the entities Study, Concept, Baseline, Objective and Requirement.

An AMN is a container/controller object identifying all the specific Model Instances and Key Value Instances (such as CAD and CAE models / results including boundary conditions and load cases) that together comprises a set of "results" for the Study. It is a network showing how Model Instances and Key Value Instances have been derived from each other.

Figure 156 shows how this kind of model networks can be used and defined consistently with simulation intent definitions. In this figure, the notion of models reuse is highlighted. Indeed, an AMN is generally associated to a simulation intent but models used in different AMNs can be used for different simulation intents. This enhances the need to aggregate several AMNs for having full traceability of Model Instances and Key Value Instances that have been derived from each other.

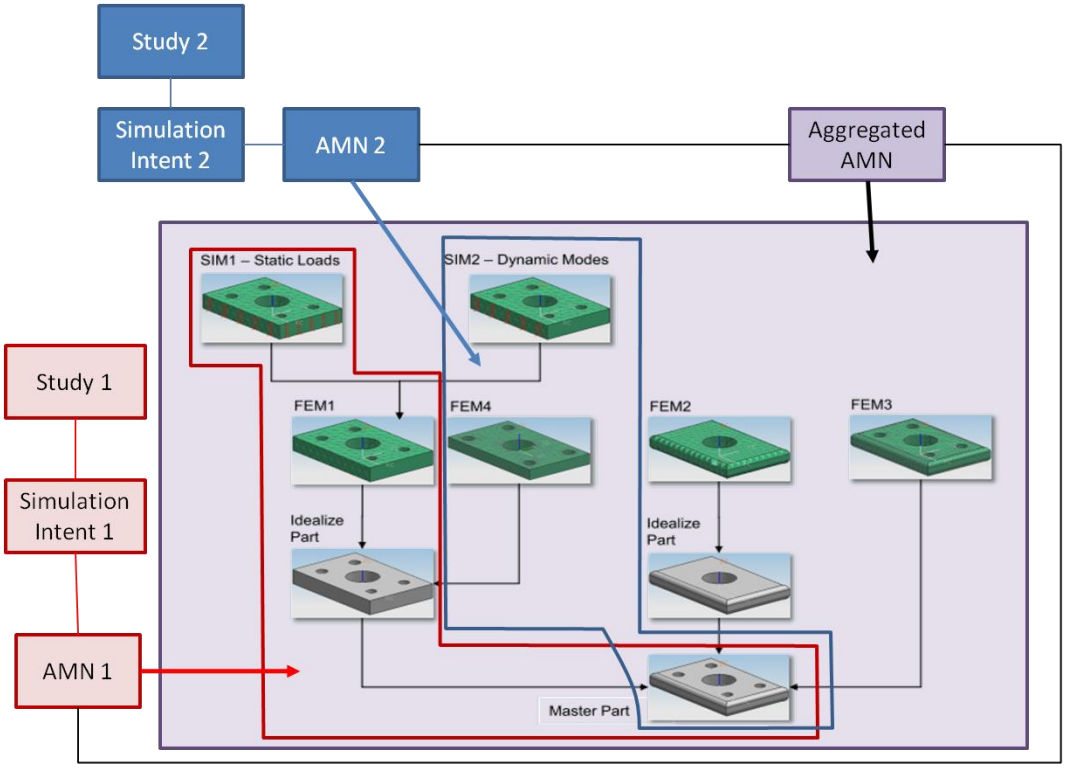


Figure 156: Example of Associative Model Networks extracts for FEA

10.3.6 Semantic data mapping for PDM/SDM interoperability

All the above mentioned concepts need to be implemented in whether CAD/CAE applications whether in EDM systems such as PDM or SDM systems.

In order to support the interoperability between heterogeneous EDM systems used by partners involved in collaborative digital integration chains the CRESCENDO project had the final objective to deliver the Behavioural Digital Aircraft (see sections 7.3 and 9.1).

As already mentioned, the BDA concept might consist in a collaborative data exchange/sharing platform for design-simulation processes and models throughout the development life cycle of aeronautical products. However, it is not expected that the BDA is a unique design environment, replacing existing ones. **Instead, the BDA has been considered as a standard data model (the BDA Business Object Model) enabling interoperability, to which existing local design environments and new services to be developed could plug.** In order to ensure the plugging of the various partners PDM/SDM systems with BDA platform and enable standardised information exchange, we propose, and based on the work from [Agostinho et al., 2010], a framework for model language independent P2P mappings.

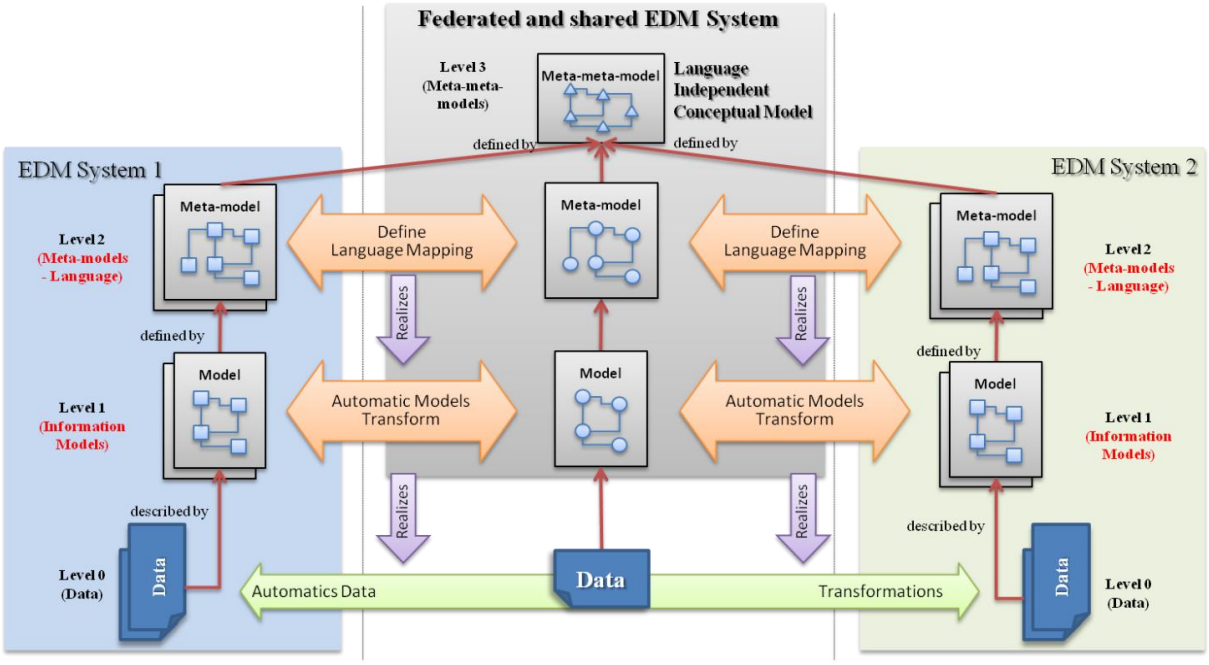


Figure 157: Framework for model language independent semantic mapping adapted from [Agostinho et al., 2010]

As illustrated on Figure 157, this framework is based on three levels of information modelling:

- Level 3 (meta-meta model): corresponding to the language independent conceptual model that might be ideally based on standardised information data models (i.e. in our case the BDA Business Object Model)
- Level 2 (meta-models): corresponding to the meta-models of the various EDM systems that need to interoperate (language dependant) and to parts of the federated conceptual model (standardised and hence language independent) that enable a language semantic mapping between the two heterogeneous meta-models of EDM system 1 and EDM system 2.
- Level 1 (models): corresponding to parts of the meta-models that will be used in the frame of a specific information exchange scenario.
- Level 0 (data): corresponding to the possible technological implementations of level 1 models in data exchange formats such as XML or RDF files.

An illustration and a demonstration of the use of such a framework for PDM-SDM interoperable information exchange are provided in section 13.2.2.3.

10.4 Conclusion

This chapter has first introduced a MBSE framework for design-analysis system integration: DASIF. DASIF could be extended and be used for any kind of design-simulation loops. In this PhD we have focused on CAD-FEA loops. The objective was to develop the digital engineering capabilities and supporting data models enabling to manage “multi-level” and “multi-domain” logical and physical DMUs enriched with assembly information in order to provide to FEA analysts and integrators the appropriate data structures and inputs to create FE assembly models.

The main approach consists in considering the DMU as a flexible product definition reference and the supporting PDM system as the main information source for designers and analysts collaborating within theses digital integration chains. The concepts of RDMU and DDMU have been introduced as

well as the conditions and scenarios for building these DDMUs. The set of concepts and solutions developed during this PhD have been introduced. Unfortunately, not all of required capabilities (derived from the functional analysis) could have been developed during this PhD. That is why Figure 158 gives an overview of what has been achieved; synthesizing the PhD contributions regarding the scientific positioning introduced in section 5.2 and derived from our industrial requirements introduced in Chapter 4.

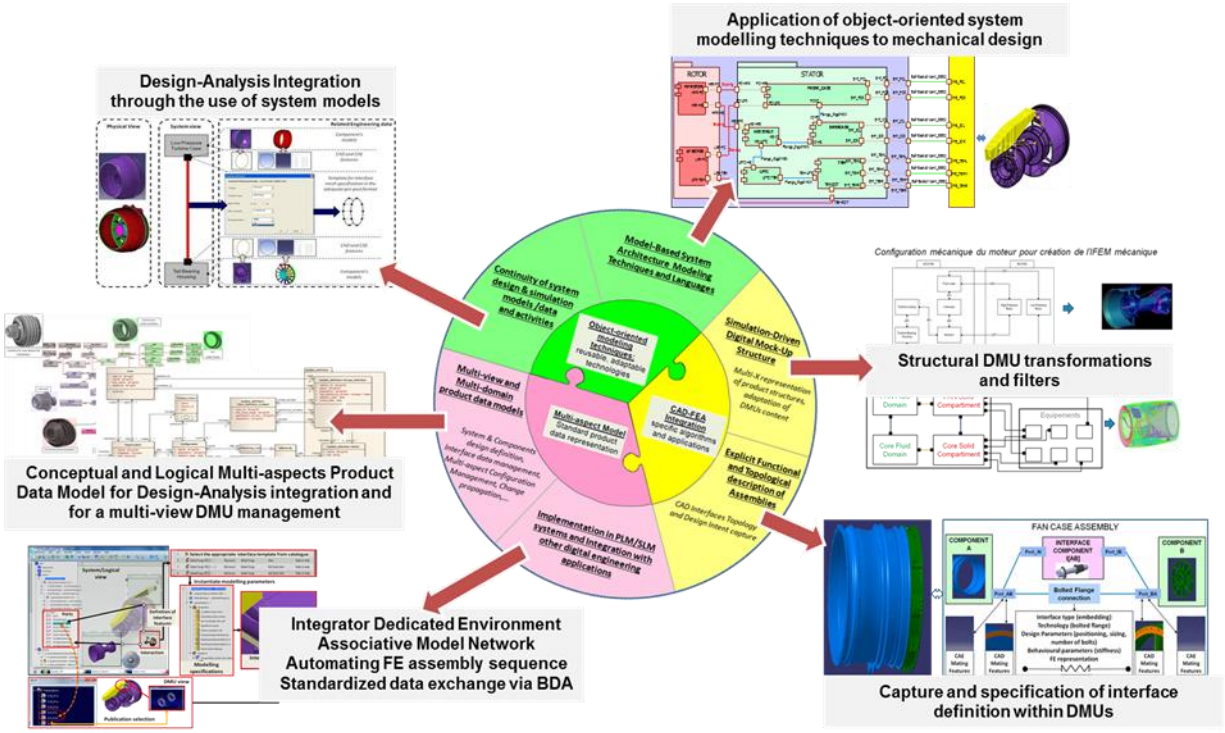


Figure 158: Synthesis of contributions regarding PhD scientific positioning

As shown in Figure 158, one of our contributions is the conceptual and logical multi-aspect product data models for Design-Analysis integration and multi-view DMU management. Extracts of this conceptual data model have been introduced in this chapter in order to clarify certain capabilities working principles. The whole DASIF conceptual data model – which is intended to be implemented whether in EDM systems data bases whether for providing standardised data structures for CAX models and related files to be exchanged – is introduced and detailed in the next chapter.

Chapter 11: Conceptual Data Model

This chapter details all the UML class diagrams describing the concepts introduced in chapter 10. In order to provide the DASIF conceptual environment introduced in section 10.1.4, we propose a multi-layered data model architecture as shown in the package diagram of Figure 159 below.

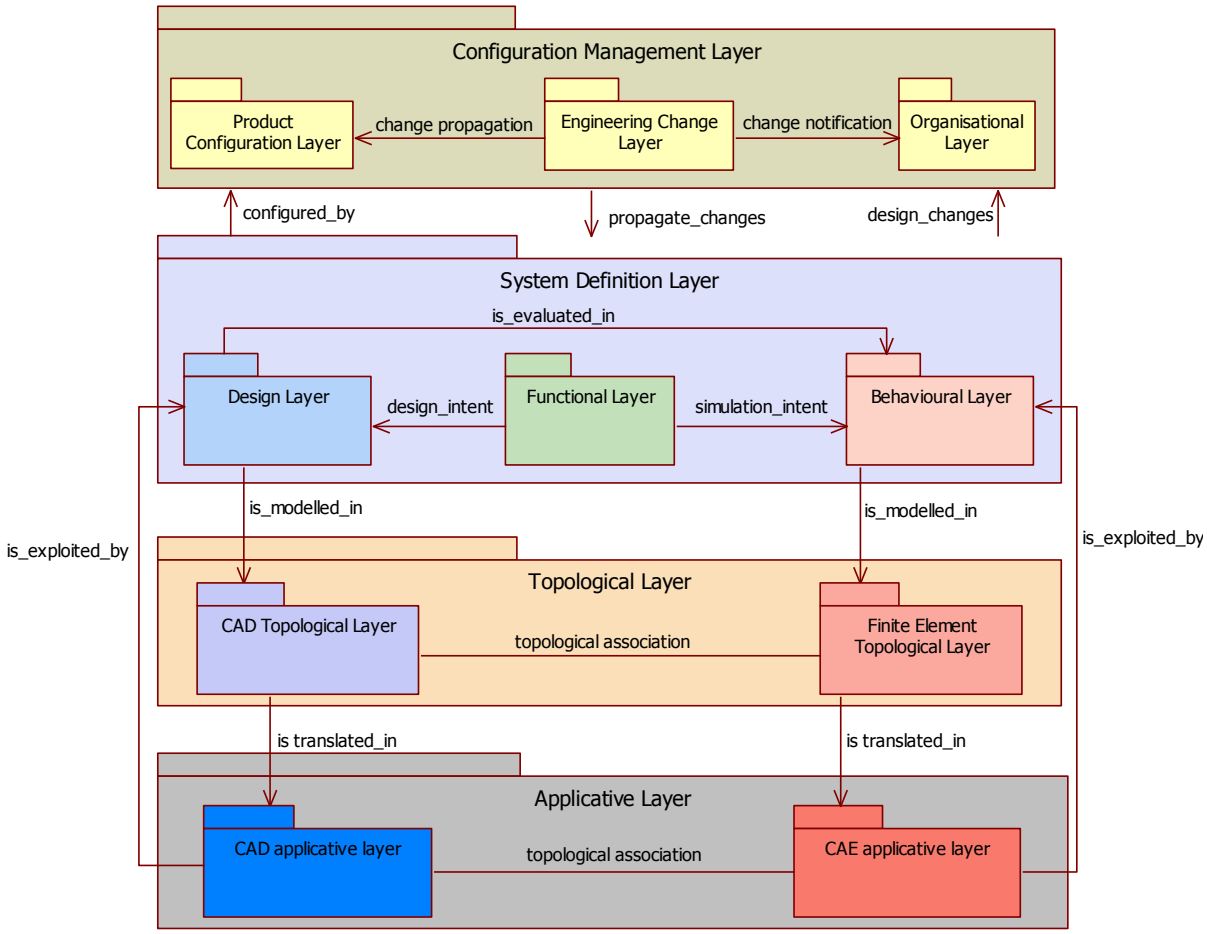


Figure 159: Conceptual architecture of DASIF data model - packages organisation

The **Configuration Management Layer** enables to manage on one hand the different product design and analysis/behavioural configurations corresponding to fit-for-purpose domain-specific analysis and on the other hand the engineering design changes and their propagation across the appropriate product configurations.

The **System Definition Layer:** this package defines the different kinds of system artefacts (e.g. parts, assemblies, interfaces) and includes all the entities permitting to define the constituents of a system/product (i.e. the system artefacts) in terms of functional, structural and behavioural aspects. Therefore it is decomposed in the three layers following layers:

- The **Functional Layer**: this layer manages the functional definition of the system artefacts; i.e. the functional system architectures and the definition of an artefact in terms of functions and related design requirements.
- The **Design Layer**: this layer manages the design definition of the system artefacts; i.e. the structural product architectures and the definition of an artefact in terms of design models and properties.
- The **Behavioural Layer**: this layer manages the behavioural definition of the system artefacts; i.e. the behavioural system architectures. It permits to manage the structure and content of

the simulation data sets (i.e. simulation models, results, behaviours description) used to verify the system artefact design definition compliance with its design requirements.

The **Topological Layer:** the topological layer provides a common data model for accessing and referencing the topological elements of whether a CAD design model whether a FE analysis model. It hence encompasses the two sub-packages: the CAD and the FE topological layers.

The **Applicative Layer:** the applicative layer should permit to use the topological description of both design and analysis models described in the topological layer, to exploit it in the appropriate CAD or CAE software authoring tools. In the applicative layer, specific mappings are necessary to translate a topological description into source files that are self-contained models in proprietary formats.

NB: the UML class diagrams have been simplified (without attributes) to provide a better readability and understanding.

11.1 System Definition Layer

11.1.1 System Artefact Definition and Taxonomy

SystemArtefact is the base abstract class for all the different kind of elements/objects participating in the definition of a system/product. It is a collector of data common to all versions of a system constituent. Therefore the instances of this class are the constituents / components of a system/product; whether they are items, structural assemblies, interfaces or even fluid domains. Below Figure 160 shows the different sub-types of SystemArtefact and also underline the three sub-types of an **ArtefactDefinition**.

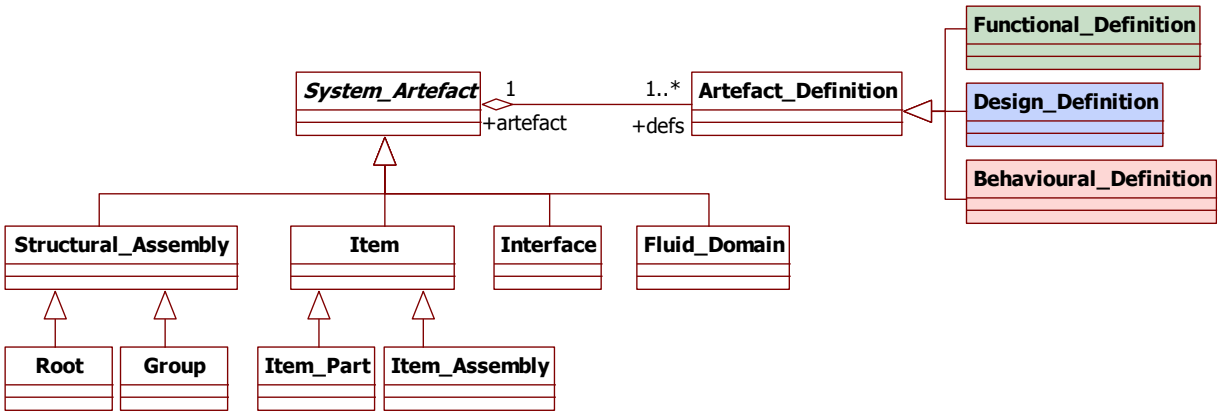


Figure 160: System Artefact Definition and Taxonomy

One SystemArtefact has 1 or more Artefact_Definitions which represents an abstract entity gathering all the characteristics and design information of a SystemArtefact. Inspired from the FBS approach from [Gero&Kannengiesser, 2004]. The three ArtefactDefinition sub-types which are **FunctionalDefinition**, **DesignDefinition** and **BehaviouralDefinition** respectively gather:

- The artefact **functional** variables: describe the teleology of the considered artefact, i.e. what it is for and what are the related technical targets (i.e. functional design requirements). It encompasses the functional building blocks permitting to describe the functional architectures of a system artefact;
- The artefact **structural or design** variables: describe the technical definition of the artefact in terms of design properties and design models as well as the components of the considered

artefact and their relationships (hierarchical and functional relationships i.e. their interfaces and interactions);

- The artefact **behavioural** variables: describe how the considered system behaves in a given operational state and permit to assess if the system fulfil the design requirements to respond to the function in this operational state. To perform this verification, a behavioural definition encompasses all the analysis definition data (analysis models, analysis results, simulation intent, etc.) .

Next section details the content of these ArtefactDefinition sub-types.

Each time a change is done on the functional, design or behavioural definition of an artefact, a new version of this **Artefact_Definition** is created. The Artefact Definition entity has a version identifier attribute considering that one Artefact Definition object represents a specific version (or revision) of this definition. Concerning the taxonomy of the different **SystemArtefact** sub-types, we have defined the entities as follows:

- **Item**: the items are the tangible constituents of a system. They represent all artefacts that make the system exist in a real world. An item intends to be the result of a manufacturing process. An item is a product that has its own characteristics (e.g. a pencil is a product and there must be different items defined for the same product (e.g. the red and the blue pencil)). An item can be whether an ItemPart or an ItemAssembly:
  - **Item_Part**: it is a single item (piece) generally considered as undismantled. However we can distinguish two kinds of ItemPart: manufacturing intermediary parts which are single items only, and parts ready to be assembled that can be whether a single item or pre-assembled items. A Part is the lowest level component. The list of ItemParts involved in a product configuration and their respective properties constitute the Bill of Materials of this configuration;
  - **Item_Assembly**: An Item_Assembly is a gathering of several ItemParts or sub-assemblies. An Assembly is a composition of its sub-assemblies and parts.
- **Structural_Assembly**: This class is necessary to represent and store the breakdown elements of a system that defines the hierarchy and the various groups of a product structure. They are not items because they only exist to structure the items and do not represent tangible constituents of a system. The StructuralAssembly entity is not abstract and can be instantiated directly. However we have defined two StructuralAssembly specific sub-types:
  - **Root**: the root is the highest level component of a product structure;
  - **Group**: They are mainly used for gathering in a same functional assembly a set of multi-instantiated parts/assemblies (e.g. a ring of bolt-nut assemblies).
- **Fluid_Domain**: it is another type of system elements that need to be considered in the definition of the product. However they are not tangible for the customer or for the manufacturing process; as a result FluidDomain elements must be integrated in product structures used in specific engineering domains, but they do not make part of the Bill of Materials;
- **Interface:** this entity intends to define/specify the physical relationships between product components. We define an Interface as the physical or theoretical boundaries where two or more system components meet and interact and where domain-specific applied rules and conventions define their interaction and related design intent. These rules and conventions concern domain-specific physical features (mechanical, electrical, thermal, etc.) semantic or functional features, and potential exchanges of information.

11.1.2 Functional, Design and Behavioural System Artefacts Definitions

The following section details each of the three sub-types of an **Artefact_Definiton**. The diagram of Figure 161 below gives the global data structure and underlines the links between functional, design and behavioural system variables.

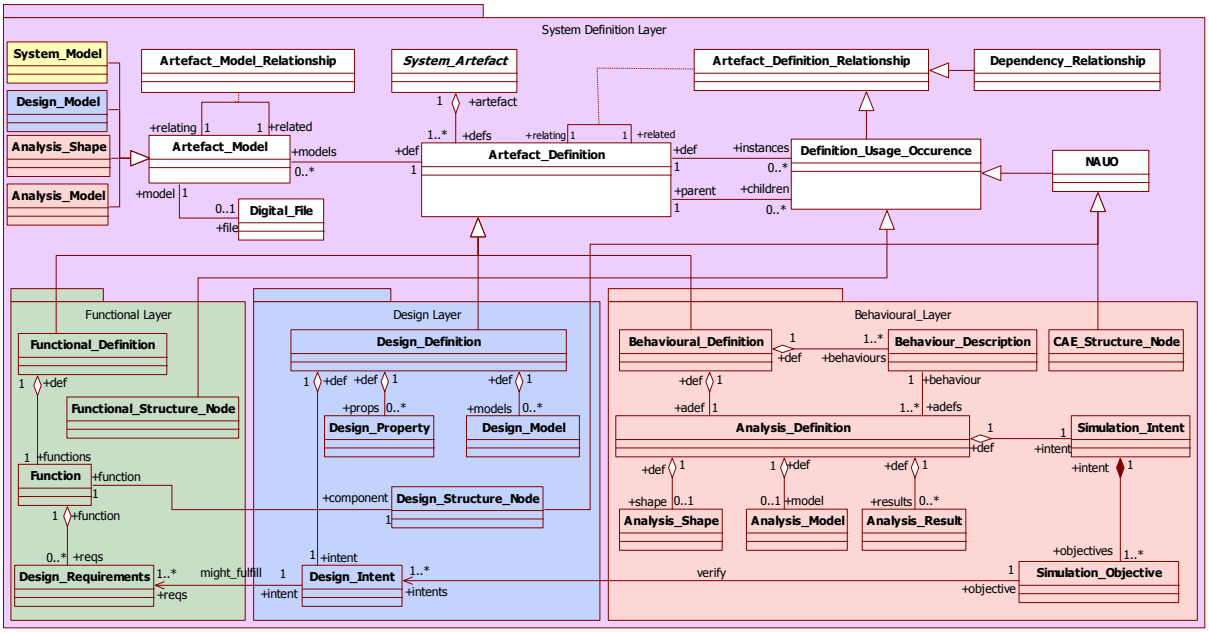


Figure 161: Functional, Design and Behavioural System Artefact Definition diagram

11.1.2.1 Functional, Structural and Behavioural product structure

An **Artefact_Definition** provides the technical definition of an artefact within a specific context (View_Definition_Context). A technical definition consists of a set of representations (models) and properties (mainly mass and material properties). An Artefact_Definition, whatever its type, can be used and instantiated in several different contexts and product views through the use of the **Definition_Usage_Occurence** entity that represents a usage occurrence of a specific artefact definition. Therefore, one Artefact_Definition has zero or more Definition_Usage_Occurence. In order to define the different kind of product structures (functional, structural, behavioural) each created instance (usage occurrence) can be attached to the definition of a parent assembly/function through the use of two sub-types of the Definition_Usage_Occurence entity (see section 11.1.3). An inheritance mechanism has been defined between Artefact_Definition and its sub-types (functional, design and behavioural definitions) in order to apply the same principle to define and manage product structures whether they are functional, structural or behavioural.

11.1.2.2 Functional, Design and Behavioural Definition core entities and relationships

A **Functional_Definition** is an aggregation of **Functions** specifying the role of the artefact in a specific context and configuration. It describes the functional architecture of a system artefact; i.e. the decomposition of an artefact function into several sub-functions. A function is an aggregation of a set of **Design_Requirements** specifying more technically and quantitatively the expected features of the artefact required to ensure the function and related expected performances (e.g. a targeted mass value of a part or assembly, the stiffness of a structural part, etc.).

A **Design_Definition** describes an artefact as an aggregation of **Design_Models** and **Design_Properties**. It also defines the **Design_Intent** of such a definition; i.e. the link with the **Design_Requirements**

that this definition might fulfil. For an assembly, the **Design_Definition** also encompass the composition of its children components. The artefact design definition of an assembly is related to the definition of its constituting sub-assemblies and parts through the use of the **NAUO** (Next Assembly Usage Occurrence) entity that represents the nodes of the design product structure.

A **Design_Model** is any kind of representation describing the structural and geometric features of the artefact. In our case the sub-type entity Nominal_CAD _Model will be instantiated for referencing the geometric model of the artefact. A **Design_Property** refers to all design features that can be useful for both designer and analyst when accessing an artefact definition. It includes features such as the mass properties or the material reference of the artefact.

A **Behavioural_Definition** is an aggregation of one Analysis_Definition and a set of Behaviours to be analysed (Behaviour_Description). An Analysis_Definition is an aggregation of the following entities:

- **Analysis_Model**: it is any kind of representation that permits to analyse the behaviour of a system artefact. In our case we will instantiate the sub-type FE_Model for referencing the finite element model used for a finite element analysis;
- **Analysis_Shape**: corresponds to the idealised CAD model used for the creation of the FE_Model of the FEA;
- **Analysis_Result**: the class permitting to access the results (raw and post-treated results) of the analysis;
- **Simulation_Intent**: this class gathers a set of simulation objectives and a set of modelling hypotheses and specifications.

The **Behaviour_Description** entity specifies the analysed behaviour in terms of the analysed physical phenomenon and their origin (specific potential product situations of life) in potential different operational phases (or **Product_State**), leading to the definition of “mission parameters” permitting to define the load cases to apply on the FE model.

11.1.2.3 Design and Analysis definition links

An important rule to notice is that an Artefact can have several Functional_Definitions, several Design_Definitions and several Behavioural_Definitions. Indeed each artefact definition instance corresponds to a specific version of the definition. Therefore the links between a functional, a design and a behavioural definition of a same artefact are not explicit. Traceable links can be established between functional, design and behavioural of a same artefact through the **Artefact_Definition_Relationship** entity and its sub-type **Dependency_Relationship**. These links can also be traced through the use of Design and Simulation Intent entities. The **Design_Intent** permits to link a Design_Definition to the set of Design_Requirements of the associated Functional_Definition. The **Simulation_Intent** (referring some Simulation_Objectives) permits to link a Behavioural_Definition to the Design_Intent of the associated Design_Definition and hence to the set of Design_Requirements the simulation intents to verify in a specific behavioural configuration.

11.1.2.4 Focus on Item Design Definition

Figure 162 provides the data structure of an item design definition. An Item_Design_Definition is a sub-type of an Artefact Design_Definition for which specific Item_Properties are defined and which is generally geometrically characterised by a Nominal_CAD_Model. Some Item_Properties are derived from this geometric model; these are the **Shape_Dependant_Properties** (volume, mass properties, etc.). Other Item_Properties can be defined independently from the geometry of the item.

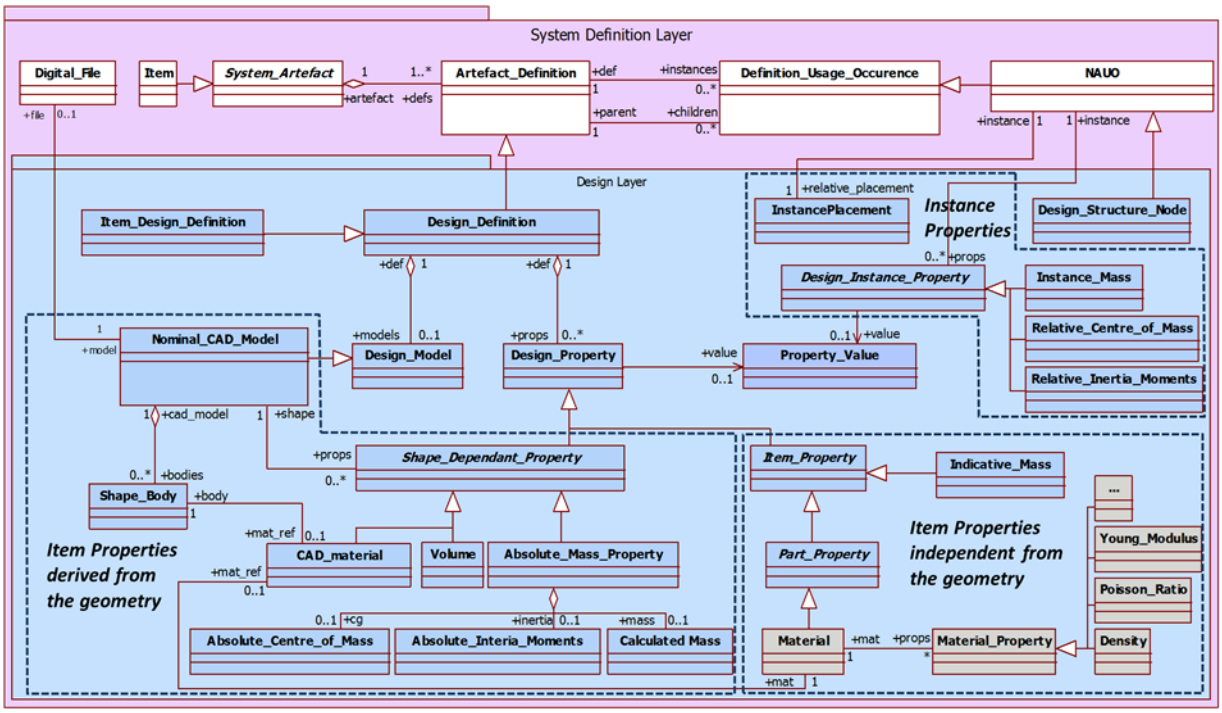


Figure 162: Global data structure of an item design definition

A CAD model can be decomposed into Shape_Bodies. It is useful to capture this level of CAD model decomposition because different material parameters can be defined for these different constituting Shape_Bodies hence impacting the mass properties of the entire item. This information is particularly required when using these material parameters for creating a FE model in the CAE pre-process.

Item_Properties must be distinguished from the Instance_Properties which are associated to the instance of the item_design_definition as used in a specific assembly. These Instance_Properties are mainly used in our context to derive relative mass properties of the instance in the coordinate system of the parent component in the assembly.

11.1.2.5 Focus on Behavioural Definition

As shown in Figure 163, in our proposed approach, an artefact Behavioural_Definition is defined as an aggregation of an Analysis_Definition (describing the analysis and the artefacts used for or derived from this analysis) and a set of Behaviour_Descriptions (defining the physical phenomena to simulate).

An Analysis_Definition is defined by the set of engineering artefacts that will be consumed or generated during the specific analysis process: a Simulation_Intent, an Analysis_Type, a Method, an Analysis_Model, an Analysis_Shape, and a set of Analysis_Results. For the CAD-FEA loop context, we have defined specific sub-types of Analysis_Shape and Analysis_Model:

- Idealised_Analysis_Shape: An Idealised_Analysis_Shape is a type of Analysis_Shape that represents a shape on which points are associated for the location of nodes in a finite element model, or a shape that is suitable for mesh generation. Mesh generation shape may include the topological information necessary to specify mesh generation curves, areas, or volumes, or may be just the bounding geometry of the mesh generation curves, areas, or volumes [ISO, 1999]. The geometry may be the Nominal_CAD_Model of a Design_Definition of the same Artefact.
- Node_Shape: A Node_shape is a type of Analysis_shape that is defined by the points of the nodes in a FEA_Model.

- FEA_Model: A FEA_Model is a collection of information that represents the finite element analysis of a product. The information includes nodes, elements, materials, properties, and groups which are combined to form a discrete mesh model of the product [ISO, 1999].

A Behaviour_Description intends to capture which physical phenomena and hence which behavioural parameters (load cases) have to be applied on the FE model to simulate these phenomena. A physical phenomenon is semantically described by the entity Phenomenon_Origin which might describe the original event of the physical phenomenon which is analysed (unbalance phenomena caused by a FAN blade-off event, a bird or ice ingestion, specific operation of the aircraft, transportation, cross-wind conditions, thermal effects, etc.). A physical phenomenon occurs in a specific Product_State (product situations of life) to which are associated a set of Usage_Status. The entity Usage_Status provides information about the operating status (working conditions) and age (new, old, end of life) of the product. It allows applying a certain coefficient/ratio to the Mission_Parameters. Product_States are defined by a set of Mission_Profiles, themselves described by a set of phases (e.g. take-off, landing, cruise, etc.) and for each of these phases a set of Mission_Parameters are defined and can be derived into Load_Cases parameters.

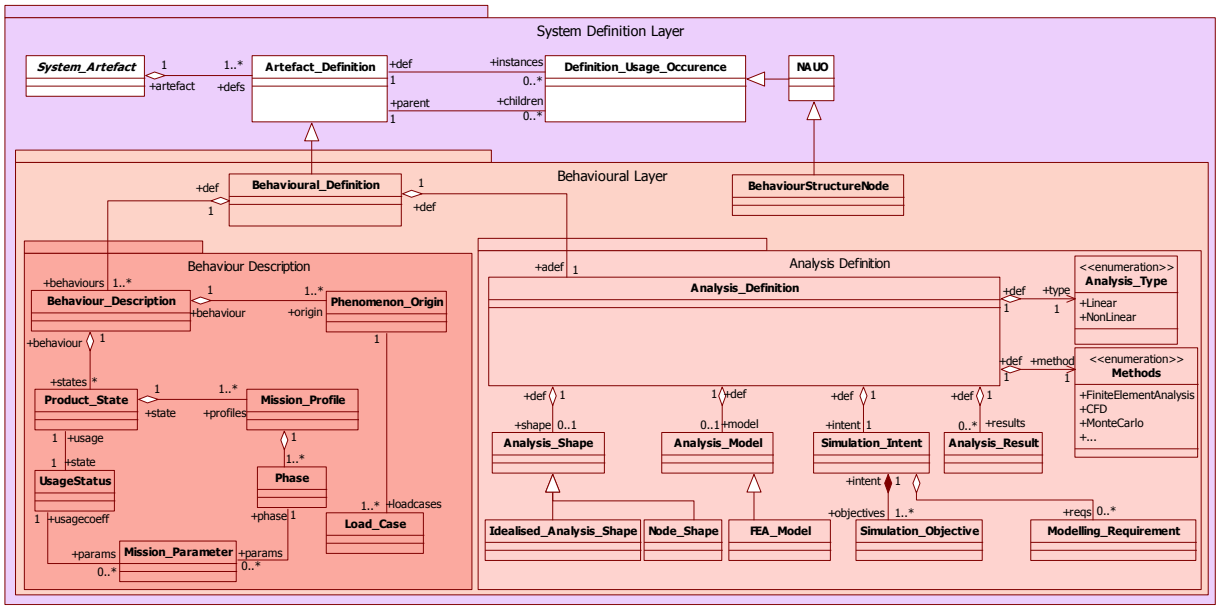


Figure 163: Global data structure of an artefact behavioural definition

11.1.3 DMU and BMU assembly structures

The data model for describing assembly structure is strongly based on the Part 44 (Product structure configuration) of the ISO-STEP standard [ISO, 2004a]. In order to define the different kind of product structures (functional, structural, behavioural) each created instance (usage occurrence) can be attached to the definition of a parent assembly/function through the use of two sub-types of the Definition_Usage_Occurrence entity:

- **Functional_Structure_Node**: It represents a single individual occurrence of an artefact Functional_Definition as used in an immediate next higher function of another artefact functional definition.
- **Next_Assembly_Usage_Occurrence (NAUO)**: it represents a single individual occurrence of an artefact Design or Behavioural Definition as used in an immediate next higher parent as-

sembly definition. The NAUOs permits to instantiate design artefacts within a product structure and within specific product configurations (through the use of effectivities), localize components in a product structure indicating their respective parent components, retrieve the definition used by components/instances and configure a product structure applying effectivities on it. The NAUO entity has two sub-types:

- **Design_Structure_Node:** these structure nodes permit to structure, configure and position the CAD assembly models in the DMU environment; since this structure node refers to an artefact definition instance, the instance properties such as the instance placement and the relative mass properties are associated to this entity.
- **Behaviour_Structure_Node:** these structure nodes permit to structure and configure the CAE simulation assembly models (e.g. integrated finite element models) in the BMU environment.

In the case of DMU or CAD assembly models and structures, the entity NAUO and its sub-types are used to define the parent-children relationships between system components. Figure 164 below shows an instance diagram of a Fan-Booster DMU assembly structure. The top level component of the product structure is created instantiating the entity Root (sub-type of Structural_Assembly). The other assembly components are created instantiating the entities Item_Assembly and Structural_Assembly. The individual parts are created instantiating the Item_Part entity. Multi-instanced parts groups such as the bolt-nut assemblies or the rotor blades group are created using the entity Group (sub-type of Stuctural_Assembly). In that case the same Design_Definition is instantiated (used) several times under the same parent assembly; only the instance placement property is evolving. The NAUO relationship is unique and relates a child component Design_Definition to the Design_Definition of its parent component in the assembly.

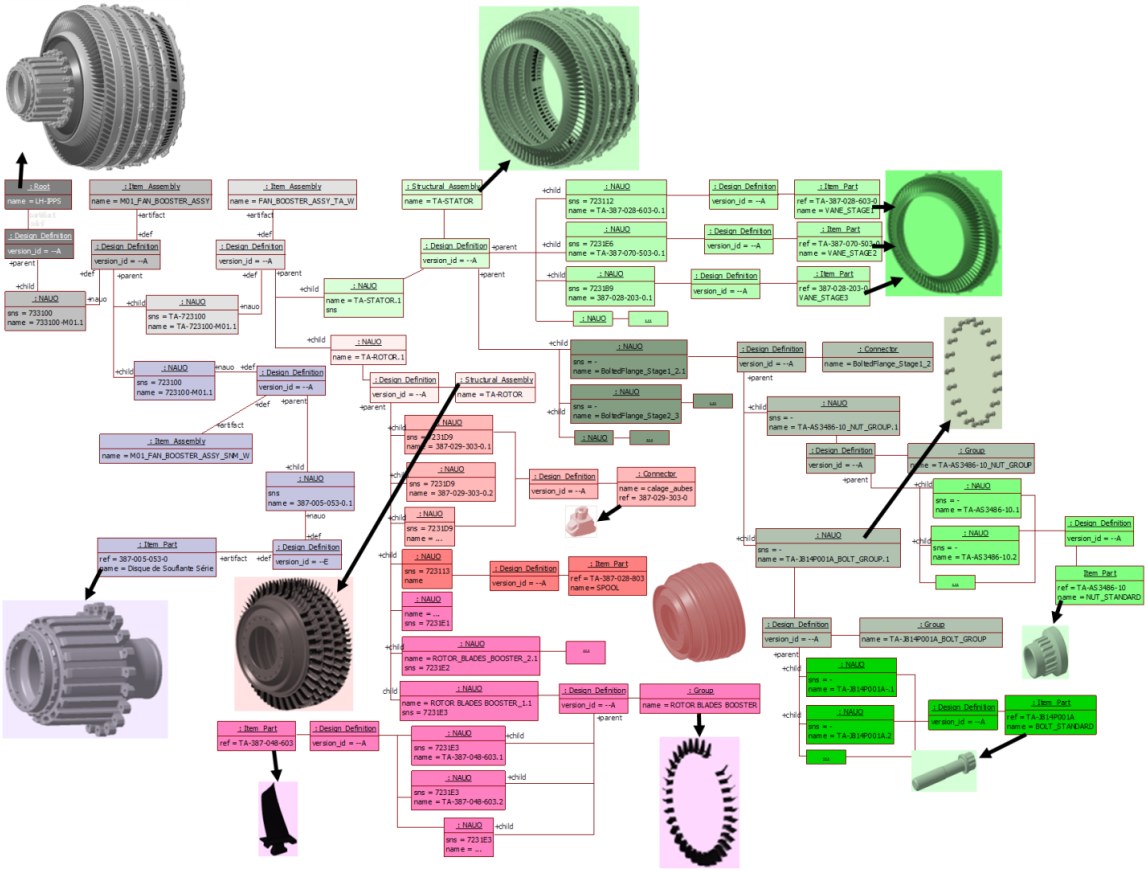


Figure 164: Instance diagram of a DMU Fan-Booster assembly structure

Since the usage of an Artefact_Definition and hence a Design_Definition is described by a Definition_Usage_Occurence, many different views of the usage can be established by varying the parent Artefact_Definition and hence creating new NAUOs of this Artefact_Definition. Thus, various Artefact_Definitions can move into other life cycle stages and usages or parts lists can be defined for any of a number of views and life cycle stages of a development process (see section 11.3.1).

11.1.4 Interface and Interaction definition

11.1.4.1 Interface taxonomy

The artefact **Interface** has a specific Design_Definition (Interface_Design_Definition) which is defined as an aggregation of a set of **Interface_CAD_Features** (partial shape elements of any geometric CAD model) and which is necessarily associated to or composed of an Interaction. Based on [Sinha et al., 2001b] and as shown in Figure 165, we have defined a taxonomy of Interactions distinguishing the Single_Domain_Interaction, the Cross_Domain_Interactions, Unintented_Interaction and Intermittent_Interaction (where the interfacing artefacts physically connect only intermittently). We have also proposed to define the following sub-types of Interface_CAD_Feature:

- **CAD_Mating_Feature**: CAD topological features specifying the interface geometrical area (surface of interaction or of non-interaction for clearances) between two interfacing components.
- **Assembly_Constraint**: CAD topological features specifying the relative position of two components in an assembly and implicitly defining the degrees of freedom between these components.
- **Kinematic_Pair**: CAD topological features specifying how the interfacing artefacts are movable with respect to one another and describes the type and/or properties of kinematic joints. One kinematic pair can point to a set of pre-defined Assembly_Constraints.

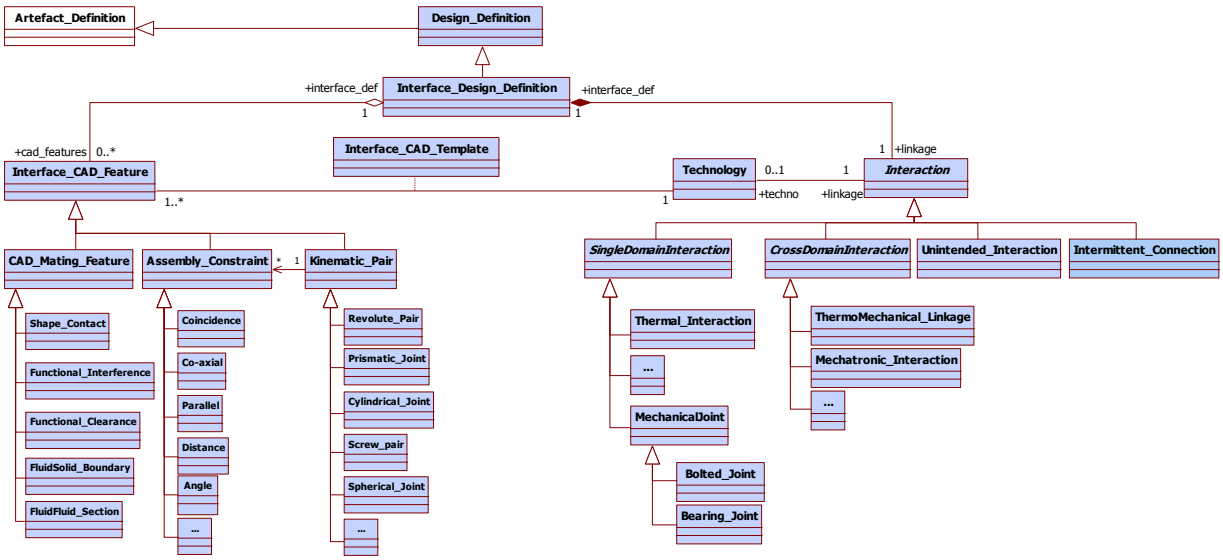


Figure 165: Proposed Interface Taxonomy

An interaction is potentially associated to a Technology (e.g. a bolted flange for a bolted joint or a roll bearing for a bearing joint). The main concept exposed here, is the idea to create **Interface_CAD_Templates** which enable to capture the set of specific Interface_CAD_Features required to define and specify a specific technological connection or Interaction (e.g. the set of planar or cylindrical contacts, clearances and/or interferences required to specify the position, the dimensions and the

alignment features of a bolted flange). Thus, the design intent of an interface (why the designer specifies an interface this way and what is the functional meaning of these mating features) is captured and re-usable.

The whole conceptual interface data model defining the functional, structural (design) and behavioural definition of an Interface artefact is provided in the following section 11.1.4.2.

11.1.4.2 Interface data model

Figure 166 illustrates the proposed data structure for defining an Interface. As other artefacts, an Interface has a functional a design and a behavioural definition and links between these three definitions are also shown in this figure.

As introduced in section 10.3.3, we propose to define an Interface according to its:

- **Functional definition**: specifying the function of the interface and the related set of design requirements (e.g. a stiffness constraint on a mechanical embedded connection or a friction ratio for a slide connection)
- **Technological definition**: specifying via NAUOs the usage of specific artefacts (and hence their appropriate definition) playing the role of connectors and enabling to ensure the interaction.
- **Kinematic definition** (specifically for mechanical joints): specifying the kinematic pair and related set of degrees of freedom (DoF) between two interfacing solid components. DoF are also used as behavioural parameters in the behavioural definition of a mechanical interface.
- **Topological definition** (specifically in the context of CAD-CAE loops): in the design layer it represents the set of partial shape elements (CAD model surfaces) defining geometrically the interface area by publishing **Shape_Feature_Publications** from CAD models and linking them through the entity **Shape_aspect_relationship** and its sub-types **Interface_CAD_Feature** and **CAD_Mating_Feature**. In the behavioural layer, it represents the corresponding set of **FE_Mating_Features** (nodes, group of nodes or elements) defining geometrically the interface area on the FE_Model of the interfacing components. It is possible to create an association between CAD and FE mating features to help analyst to retrieve the right location of an interface and to better understand the way a re-used assembly FE model has been created.
- **Behavioural Definition**: The **Interface_Behavioural_Definition** is, like another behavioural definition, related to a specific CAE analysis. It gathers the FE topological definition, a set of behavioural parameters and potentially a set of **Boundary_Conditions** to apply on this interface to run the simulation. Moreover, the Interface_Behavioural_Definition is potentially represented by a **FE_Link_Model** specifying physical connections between two **FE_Models**.

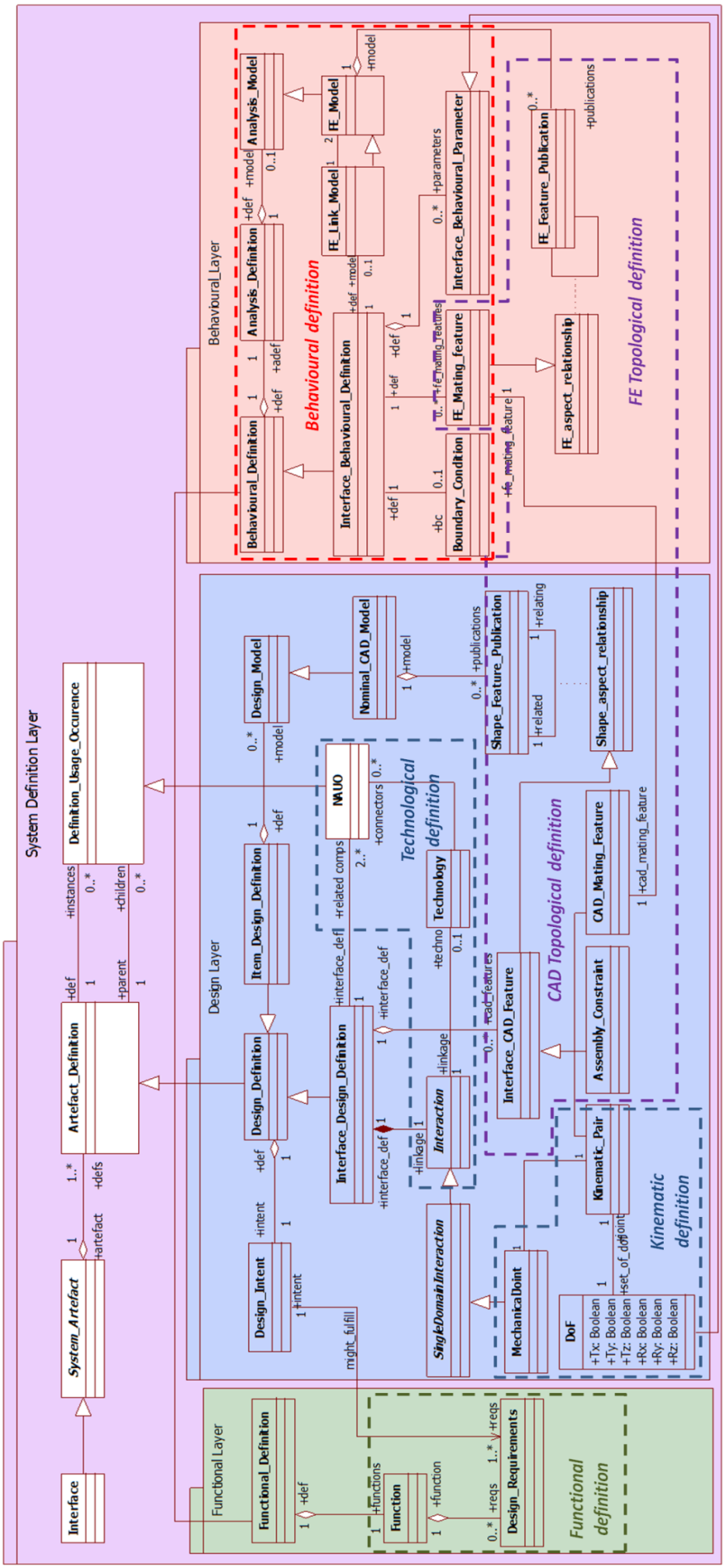


Figure 166: System Interface Data Model

11.1.5 System Models

This part of the DASIF conceptual data model has been defined in order to define the data structure of a system model and to define the possible associations with the engineering artefacts and their definitions used for configuring, with an object-oriented and systemic formalism, the DMU and BMU assembly models. As shown below in Figure 167 and based on SysML internal block and parametric diagrams [Friedenthal et al., 2011], a **System_Model** is an aggregation of **System_Blocks**, **Ports** and **Connectors**.

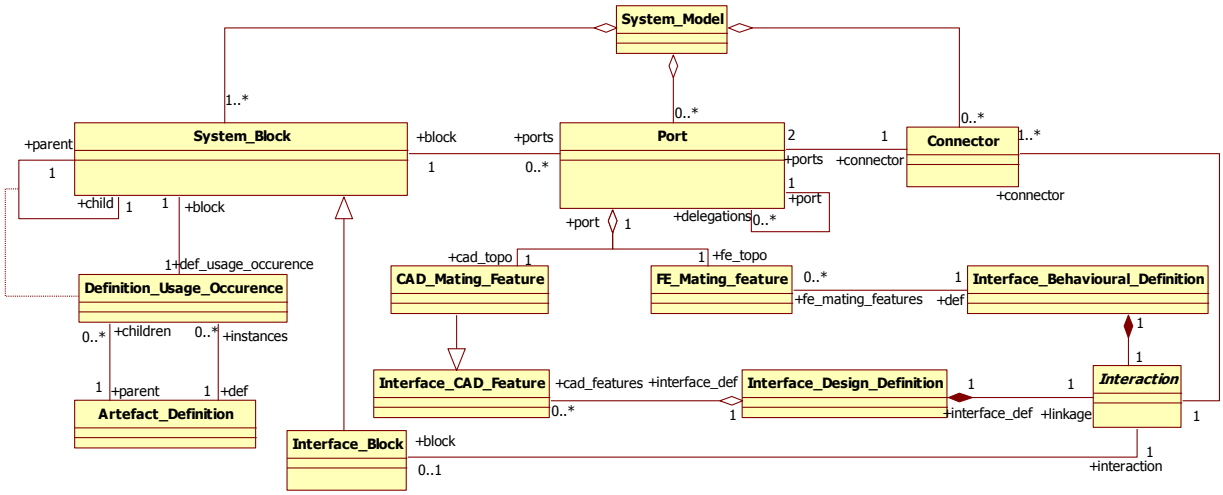


Figure 167: Data model for defining DMU and BMU system models

System_Blocks are used to reference one and only one usage of any Artefact_Definition in a specific system view. Since it references Definition_Usage_Occurences, it is possible to access to this definition and hence to the required artefact models and properties.

To one System_Block several Ports can be defined as the different interface area associated to the referenced component usage.

Connectors enable to connect two System_Blocks via Ports. It necessarily references an Interaction with its associated Interface_Design_Definition and Interface_Behavioural_Definition. Ports can reference a set of CAD_Mating_Features and FE_Mating_Feature. If a set of CAD_Mating_Features and FE_Mating_Feature are associated to the same port, they are implicitly associated for specifying the same interface area. As mentioned in section 10.3.4.2**, Interface_Blocks** enable to describe an interface as a system constituted of sub-systems or components. An Interface_Block is also necessarily associated to an Interaction with its related Interface_Design_Definition and Interface_Behavioural_Definition.

11.2 From Artefact System Definition Layer to Topological Layer

Each Artefact_Definition is potentially described by a set of numerical models; the Artefact_Models. The entity **Digital_File** represents the numerical data file containing the information describing the Artefact_Model. An artefact model has potentially one **Topological_Representation** if it is a geometric model. In the case of CAD and FEA models this Topological_Representation is whether a Shape_Representation (for CAD models) whether a FE Representation (for FE models) as illustrated in Figure 168.

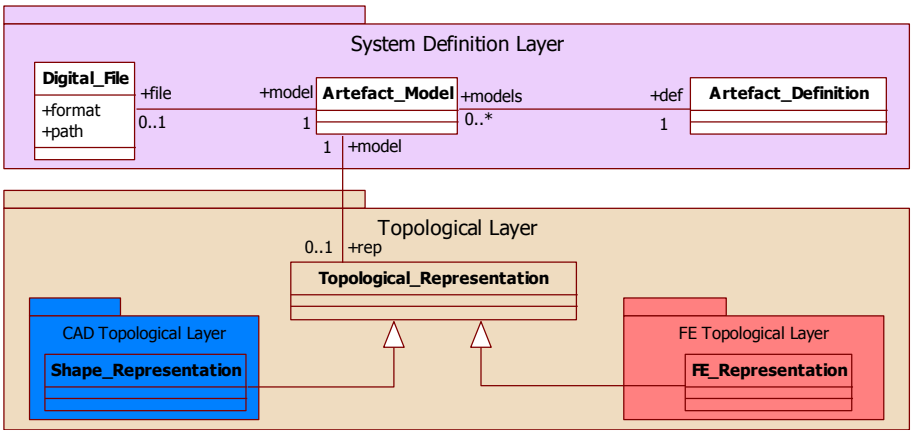


Figure 168: Link between System Definition and Topological Layers

The **Shape_Representation** data structure is based on the Part 42 (Geometric and topological representation) of the ISO-STEP standard [ISO, 2003a]. The **FE_Representation** data structure is based on the Part 52 (Mesh-based topology) [ISO, 2011b] and Part 104 (Finite element analysis) [ISO, 2000a] of the ISO-STEP standard. Figure 169 and Figure 170 provides the data models describing how the system definition layer and the topological layer are linked for respectively representing the topology of CAD models (such as the **Nominal_CAD_Model** and the various sub-types of **Analysis_Shapes**) and FEA models.

11.2.1 CAD topological layer

A **Shape_Representation** has one specific type of representation (Brep model, wireframe 2D, surface model, etc.). A Shape_Representation, according to its type of representation, is described by a set of **Geometric_Representation_Items** (which are the concrete topological elements) permitting to describe the geometry. Each Geometric_Representation_Item is defined in a specific **Geometric_Representation_Context** (identifying the coordinate space in which it is defined) and positioned for instance in a 3D coordinate system defined by the entity **Axis2_placement_3D** (sub-type of the **Placement** entity which is a specific type of Geometric_Representation_Item). The relative positioning of different Shape_Representations in an assembly can be captured through the use of the Shape**_representation_relationship_with_transformation** referencing a **Cartesian_Transformation_Operator** (position matrix). This information can be passed on to the system definition layer through the **Instance_Placement** entity, enabling to get this information without loading the CAD model in a CAD modeller.

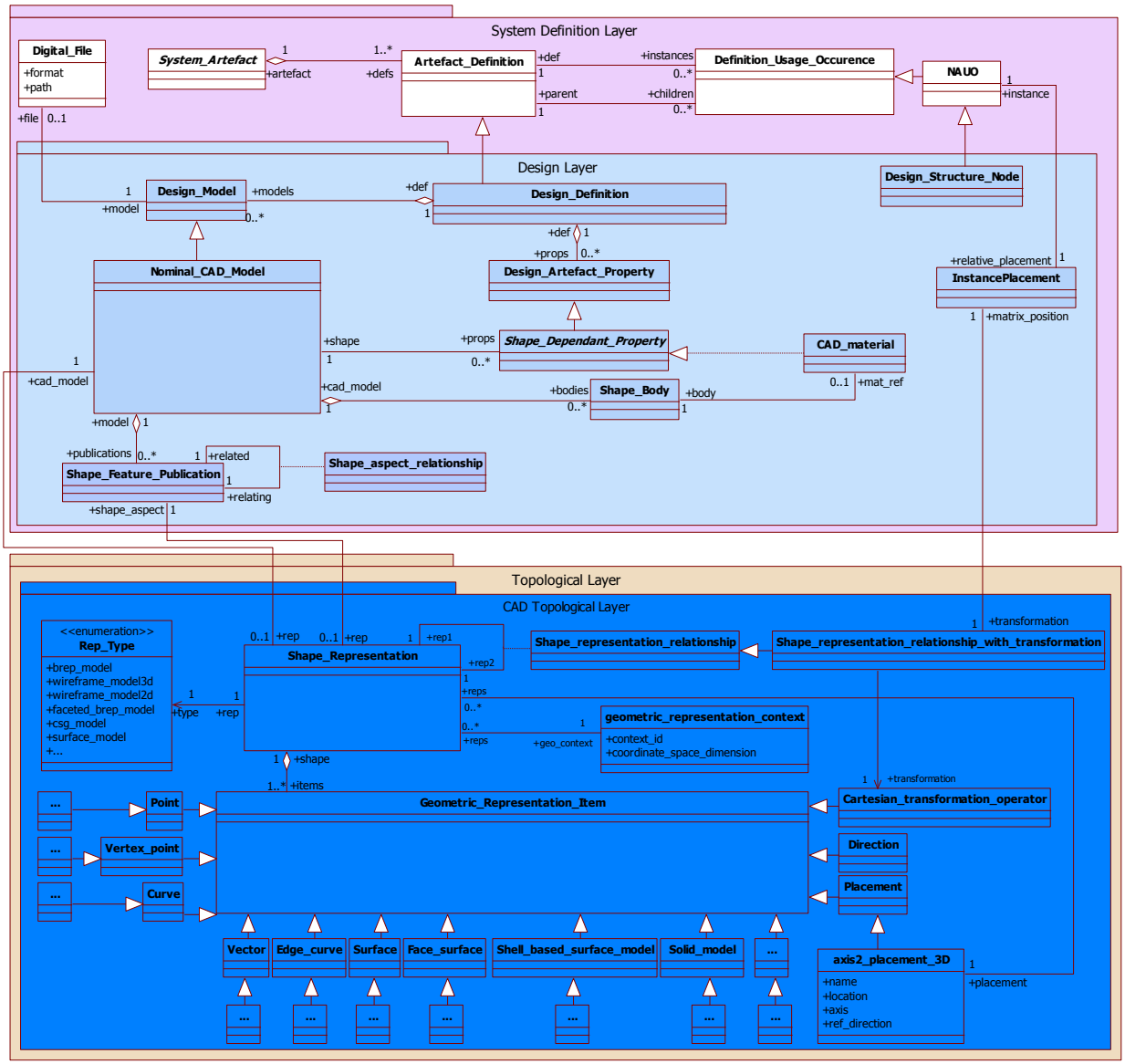


Figure 169: Links between Design Layer and Topological Layer

11.2.2 FE topological Layer and links with the CAD topology

A **FEA_Model** is an aggregation of a **FE_Representation** with a set of **Load_Representations**. A Load_Representation represents the numerical values of load cases to apply to the FE Representation to run the simulation and assess the expected behavioural phenomenon. A FE_Representation is an aggregation of **FE_Nodes** and **FE_Elements**. A FE_Node is a discretization point for the field variables of the FEA_Model. A FE_Element is the basic building block of a FEA_Model. It defines the mathematical relationship between the finite element Nodes. An Element shall be one of the following in ISO10303-104 [ISO, 2000a]: Curve_element, Directionally_explicit_element, Explicit_element, Point_element, Substructure_element, Surface_element, Volume_element. We have categorised these types of elements according to their respective dimensions (0D, 1D, 2D or 3D). The **Node_Placement** entity specifies the location and hence the coordinates of the Node with respect to the founding **FE_Placement_Coordinate_System**. The FE_Placement_Coordinate_System entity specifies the coordinate system for the analysis variables at the Node. Groups of Nodes can be created and related together (specifically useful in assemblies to define a FE connection between two mating groups of nodes).

Topological associations can be made between the CAD topological elements of a nominal or idealised CAD model and the FE topological elements of a FEA model. These “fine-grain” associations are captured by the entities **Node_Shape_Relationship** and **Element_Shape_Relationship** for linking Geometric_Representation_Items of a Shape_representation with respectively FE_Nodes and FE_Elements.

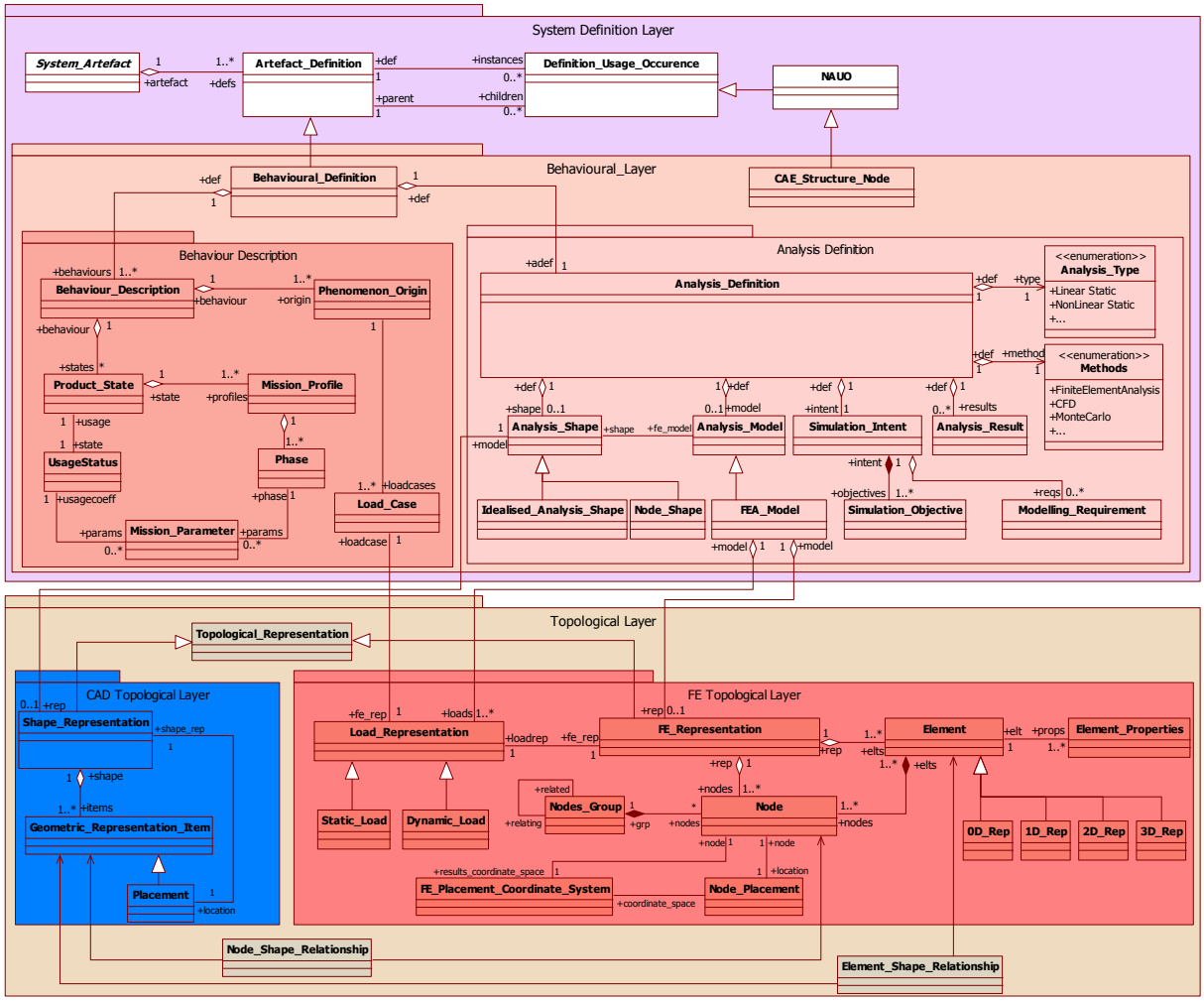


Figure 170: Links between Behavioural Layer and Topological Layer

11.3 Configuration Management Layer

The **Configuration Management Layer** enables to manage on one hand the different product design and analysis/behavioural configurations corresponding to fit-for-purpose domain-specific analysis and on the other hand the engineering design changes and their propagation across the appropriate product configurations. Therefore this layer is sub-divided in three sub-packages which are:

- The **Product Configuration layer:** this package encompasses all the product manufacturing and design configurations as well as the configurations effectivities assignments to the relevant component definitions;
- The **Engineering Change layer:** this package defines the Engineering Change object that permit to capture the evolutions of the technical definition of components. This object might identify the nature of the design change(s) and enable the access to the data or meta-data that have evolved. This object is also configured since a design change might concern one or several product configurations;

- The **Organisational layer:** this package manages the organisations (companies, departments and business units) involved in the product development process, as well as their relationships (e.g. hierarchical or supplier-customer relationships). One organisation is composed of employees which are potential users of DASIF. This package also manages the assignment of product data or meta- data to some users (responsible of a design definition, responsible of a design change, responsible of an analysis, etc.).

The Configuration Management layer has several relationships with the system definition layer. It defines the "effective usage" s of artefact definitions in the appropriate product configurations. This layer also captures the various design changes / evolutions coming from the system definition layer and enables to propagate these changes in the appropriate product configurations and views.

11.3.1 Multi-view and design-oriented configuration management

We consider the business/marketing definition of a product: a thing (a tangible good in our case) that can be offered to a market that might satisfy a want or need and which is the result of a manufacturing process. Therefore, a **Product_Class** is the identification of a set of similar products to be offered to the market. It rather specifies an aero-engine or the related aircraft application. That is why there can be hierarchal links between Product_Classes (e.g. The LEAP-X engine will be the child of rather an A320 Neo or 737MAX aircraft).

A **Configuration** is a product variant (or Configuration Item in ISO10303-44), i.e., the identification of a particular variation of a Product_Class. A product variant is defined with respect to the product class (product concept in part44), i.e., the class of similar products of which it is a member. We distinguish three different kinds of configurations according to what is proposed in section 10.3.2.2:

- **Master_Configuration**: it is a product configuration that is established each time it is necessary to agree on a reference which is the basis for identifying further configurations and related product views. Based on pre-defined validation plan, and on the experience of previous projects, the derived "design discipline configurations";
- **Design_Discipline_Configuration:** These configurations correspond to variants of a product used in specific application domains related to the design /development activities. These configurations are not necessarily intended to be delivered to a customer organisation as a manufacturable item, but they can be and are generally based on or associated to a manufacturable configuration. They are variation of a product class, or its discrete portions that are treated as a single unit in the configuration management process. This can be for example a mechanical configuration of the aero-engine that decomposes the engine with a rotor-stator decomposition in view to perform mechanical-specific analyses;
- **Manufacturable_Configuration:** These configurations define a manufacturable end item, or something that is conceived and expected as such. Depending on the kind of industry and products, a product_concept (Product_Class) might be offered to the customers in one or many different manufacturable configurations. If the product concept is offered in different configurations, each of these configurations is a member of the class of products defined by this product concept.

Dependency links between a Master_Configurations, Design_Discipline_Configuration and Manufacturable_Configurations can be defined by instantiating the entity **Configuration_Relationship.** These relationships enable to trace from which configuration another configuration has been derived.

This implies that there are master-slave links enabling change propagation between Definition_Usage_Occurences used in these configurations (see next section 11.3.2).

Figure 171 shows the data model enabling to build, capture, modify and make evolve the various product configurations and how they are linked to their constituting elements (usage occurrences of Artefact_Definitions) through the concept of “effectivity”.

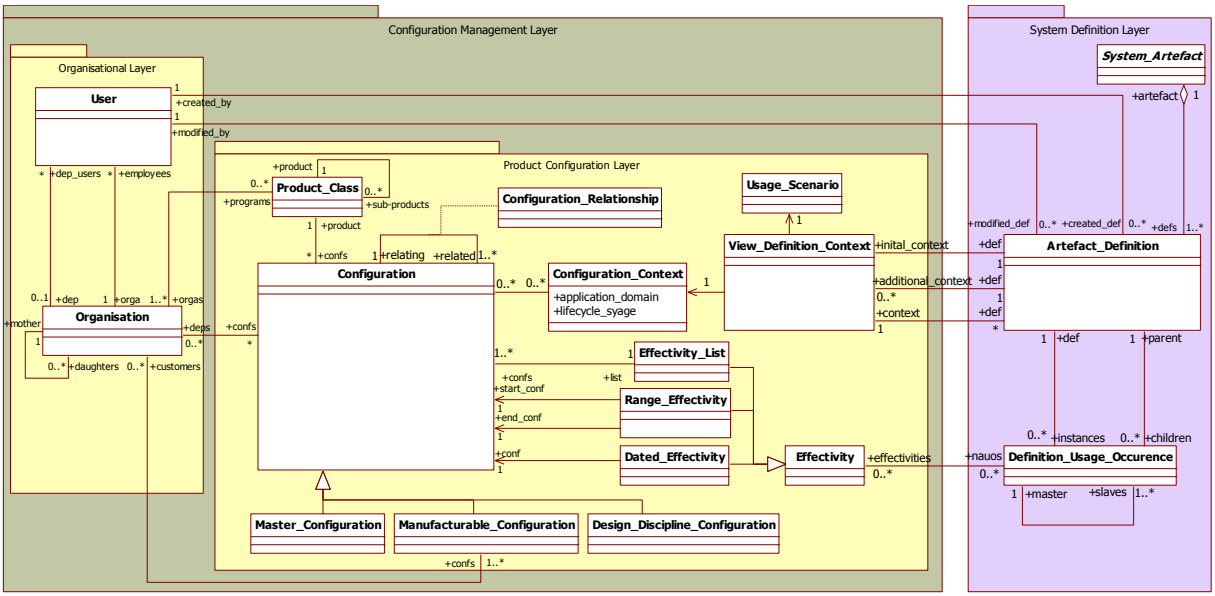


Figure 171: Product configuration data model - links between configurations and system definitions

Effectivity is the key concept for linking the system Artefact_Definitions to the appropriate product configurations in which they are valid. An Effectivity is the identification of a domain of applicability for product data; i.e. defining the valid use of the product data to which the effectivity is assigned. Effectivity allows control of the constituent parts (instances of Artefact_Definitions) that should be used to build the physical instances of a product configuration. The composition of the product configurations for planned units of manufacture may be controlled for a given time period, lot, or serial number range. This is managed using **Dated_Effectivity**, **Range_Effectivity**, or **Effectivity_List**. Instances of Effectivity may be applied to any Artefact_Definition instances or usage occurrences (e.g. NAUO).

The **View_Definition_Context** defines the context in which the design definition of an artefact (properties, documents, structures, etc.) is valid. A View_Definition_Context is the grouping of a **Configuration_Context** (defining the application domain(s) and the life cycle stage(s) in which an artefact definition is valid) and a **Usage_Scenario** (representing the use case(s) of the artefact definition and related representations). Indeed, within a same discipline or application domain and a same life cycle stage, several different use cases of a same product representation can exist (e.g. simulate the impacts of a physical phenomena, design in context, specify components interfaces, exchange of CAD data, etc.).

An Artefact_Definition is a characterization of an artefact definition version, relevant in one or more context application domains, one or more life cycle stages and for potentially several use cases. An Artefact_Definition is a collector of the properties that characterize the System_Artefact in the initial_context and additional_contexts.

The organisational layer – here encompassing the **User** and **Organisation** entities – enable first to define the structure of an Organisation around a product development program or project (identified

through the use of Product_Class) and to assign Users of the environment to the different Organisations involved in the program. An Organisation can be assigned to specific product Configurations to develop or to produce. This layer also enables, according to these assignments, to define specific roles and access rights to users. Users are also assigned to the Artefact_Definition they have created or modified and to subscribe to several definitions to be notified of their evolutions.

11.3.2 Engineering Design Change Management

Figure 172 below provides the proposed data model for engineering change management inspired from principles defined in ISO10303-214 [ISO, 2000b].

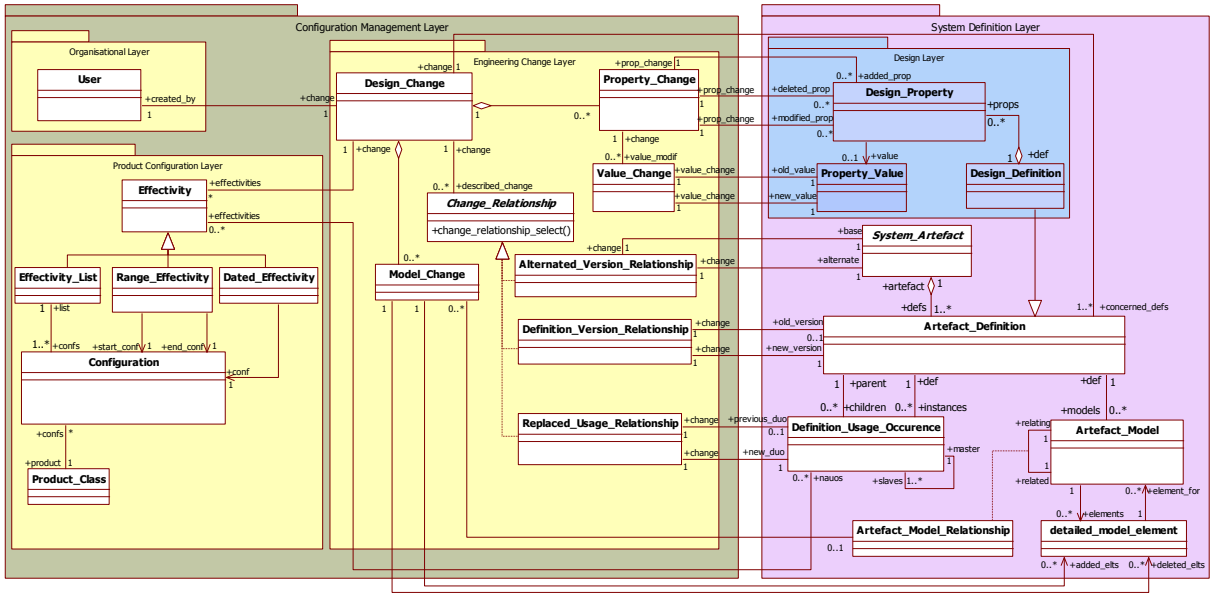


Figure 172: Engineering change management data model - links between changes, configurations and system definitions

A **Design_Change** is a mechanism to collect the **Model_Change** objects and the **Property_Change** objects that describe the differences between the two objects referenced by the described relationship.

A **Model_Change** is a mechanism to describe the differences between two objects concerning the models describing these objects. Model_Change mechanisms enable first to specify the Artefact_Model_Relationship that identifies the original (relating) model and the changed (related) model and secondly to access to the Detailed_Model_Elements (topological elements such as Geometric_Representation_Items for Shape_Representations) that have been added or deleted to the related or relating Artefact_Model. For a CAD model, a Detailed_Model_Element is a single geometric element of a geometric model. In ISO10303-214 a Detailed_Model_Element can also be a Kinematic link, a Template instance, a Kinematic structure or a Kinematic joint (see [ISO, 2000b]).

A **Property_Change** is a mechanism to describe the differences between two objects concerning the properties of these objects. This entity capture all added, deleted or modified Design_Properties of an Artefact Design_Definition. If modified properties have a **Property_Value**, the Property_Change object also references the **Value_Change** object enabling that captures and traces the evolution of a certain Property_Value.

A **Change_Relationship** provides the identification of the successive modification whether it is a definition version, a model revision, a new design alternative, or a modification of a Definition_Usage_Occurrence (new one, removal).

Each created Design_Change object references the User that has created this Design_Change. Each Design_Change is associated to a list of Effectivities referencing the various Configurations in which the change might be propagated. One Design_Change object can concern and impact several Artefact_Definitions. And one Artefact_Definitions could have been impacted or concerned by several Design_Change objects. It is hence possible to trace all Design_Change objects created for one Artefact_Definition.

11.4 Conclusion

The conceptual data model introduced in previous sections provides the required informational model to organise and manage the engineering artefacts used in the frame of the DASIF framework. It intends to support the management of product data generated and/or consumed project activities and actors involved in CAD-CAE digital integration chains. Although class and objects behaviours as well as relationships between entities are provided in the documentation, attributes and operations of classes are not displayed in this conceptual data model.

This data model has several objectives:

- To be enriched with required attributes and operations/methods according to the specific business requirements of the environment where it has to be implemented;
- To be implemented in a relational database and its client applications constituting the product and simulation data management systems supporting DASIF;
- To specify the data structure of exchanged CAD and CAE data files (topological and applicative layers);
- To contribute to the enrichment of the BDA object model with the proposed concepts and services so that they can potentially be implemented within the BDA hub platform in order to support collaborative data exchange within CAD-CAE digital integration chains.

Not all parts of this conceptual data model have been implemented in the prototypes introduced in next chapters. Next part introduces the different implementations of these concepts performed in the frame of this PhD.

PART IV: IMPLEMENTATION AND DEMONSTRATIONS OF CONCEPTS

This part of the dissertation aims at demonstrating that the concepts proposed in Part III can be implemented and that they can fulfil the industrial requirements that this PhD addresses. This part is made of four chapters.

In order to develop and implement these concepts, a PDM system mock-up has been proposed and developed at Snecma. One important objective of this prototype is also to allow testing of related concepts and identify difficulties related to implementation. This prototype and related achievements are presented in chapter 12.

Moreover, within the CRESECNDO project we have also contributed in assessing the maturity of existing commercial application to implement our concepts. Indeed we have developed, in collaboration with PLM/SLM software vendors, prototypes intending to prove the feasibility of implementing such concepts in PLM and SLM platforms. We hence developed a demonstration scenario of a “To-Be” integration process: the **product integration scenario**. This scenario and the related demonstrators are fully explained in Chapter 13. Reminding that one of the major objectives of CRESCENDO is to provide the Behavioural Digital Aircraft (BDA) (see sections 3.4 and 7.3), we have also contributed in defining some parts of the BDA Business Object Model (BDA BOM). The BDA BOM represents the data model capturing information used or provided by the BDA functions and services. Our contribution to the development of the BDA BOM is also addressed in Chapter 13.

Chapter 14 is dedicated to the PhD results validation; i.e. to the assessment of the capabilities developed for the DASIF prototype and for the Product Integration scenario.

Finally Chapter 15 is the general conclusion of this PhD. It provides a general synthesis of our research works and contributions but also new open perspectives for future research works and developments.

Chapter 12: DASIF prototype development and exploitation

12.1 Prototype architecture

The developed prototype has been built on a classical server-clients environment (see Figure 173). The server is a relational data base system host on a local machine. The clients are on one side the DASIF PDM system and on the other side the CAD modeller application (here CATIA V5). In the future other clients can be considered such as CAE applications and the DASIF SDM system.

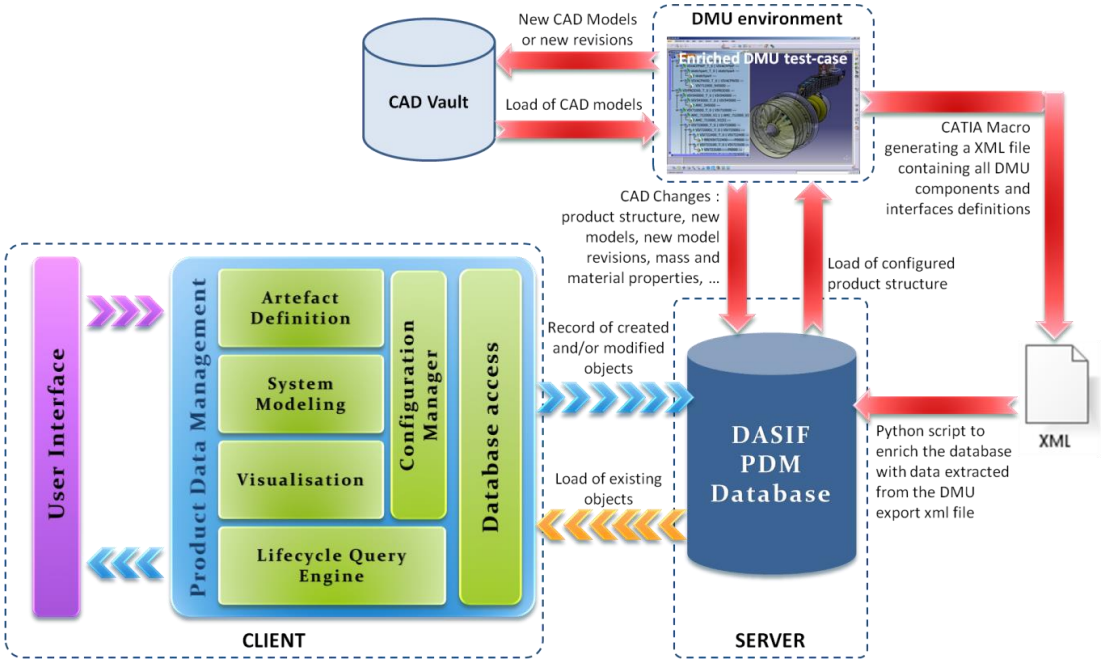


Figure 173: first DASIF prototype architecture

A specific CAD vault has been set-up to save all the CAD models so that the DASIF PDM data base is not obstructed and slowed by loading all these voluminous CAD data and so that they can be accessed and exploited by other data base systems. The DASIF relational data base system implements the logical data model introduced in next section. It aims at storing and managing product data and meta-data. The data base is enriched whether by creating and saving new objects in the PDM client application or by extracting data directly from the CAD modeller based on existing DMUs.

For this first prototype, the data base has been enriched with an XML file generated with a dedicated macro in CATIA V5 (see section 12.3.2). The DMU that served for enriching the data base has been configured for the demonstration purpose and enriched with interface mating features and the generated XML file also includes mechanical interaction specifications referencing these mating features.

The PDM client application is structured in 6 applicative modules:

- The data base access module: includes applicative functions enabling to connect the client to the data base, to enrich/update the data base with new created/modified objects and to load the objects saved in the data base so that they can be accessed and displayed within the other applicative modules.
- The lifecycle query engine: this interface permits to search and query objects in the data base through the use of simple queries.

- The configuration manager: this module permits to create, tovisualise, to modify product configurations structures, to compare configurations (visualise modifications) and to generate specific business configurations and map them to the referential configurations.
- The artefact definition module: enables to display and to modify artefact attributes, mainly the product components attributes (name, item identifier, functional item relationship, mass properties, position, material reference, etc.)
- The system modelling module: enables to visualise and manage product configurations as systemic representations (blocks, embedded blocks, connectors and ports).
- The viewer (visualisation module): permits to visualise CAD model in standardised visualisation formats (JT, STEP).

An overview of the developed graphical user interfaces of some of these modules is provided in section 12.4.

12.2 Logical data model for implementation

This section provides the UML class diagram of the DASIF data base as it was implemented for the prototype developed at Snecma. This data model is similar to the conceptual data model introduced in Chapter 11, but it has been simplified to enable an easier and faster implementation. Simplifications consist in factorising classes and relationships and in removing some abstraction levels (inheritence and/or specialisation relationships). No additional documentation is provided in this section since the explanations are the same than the ones provided in Chapter 11. The attributes and methods/operations of the classes are documented in the corresponding generated data base code provided by appendix X and Table 19.

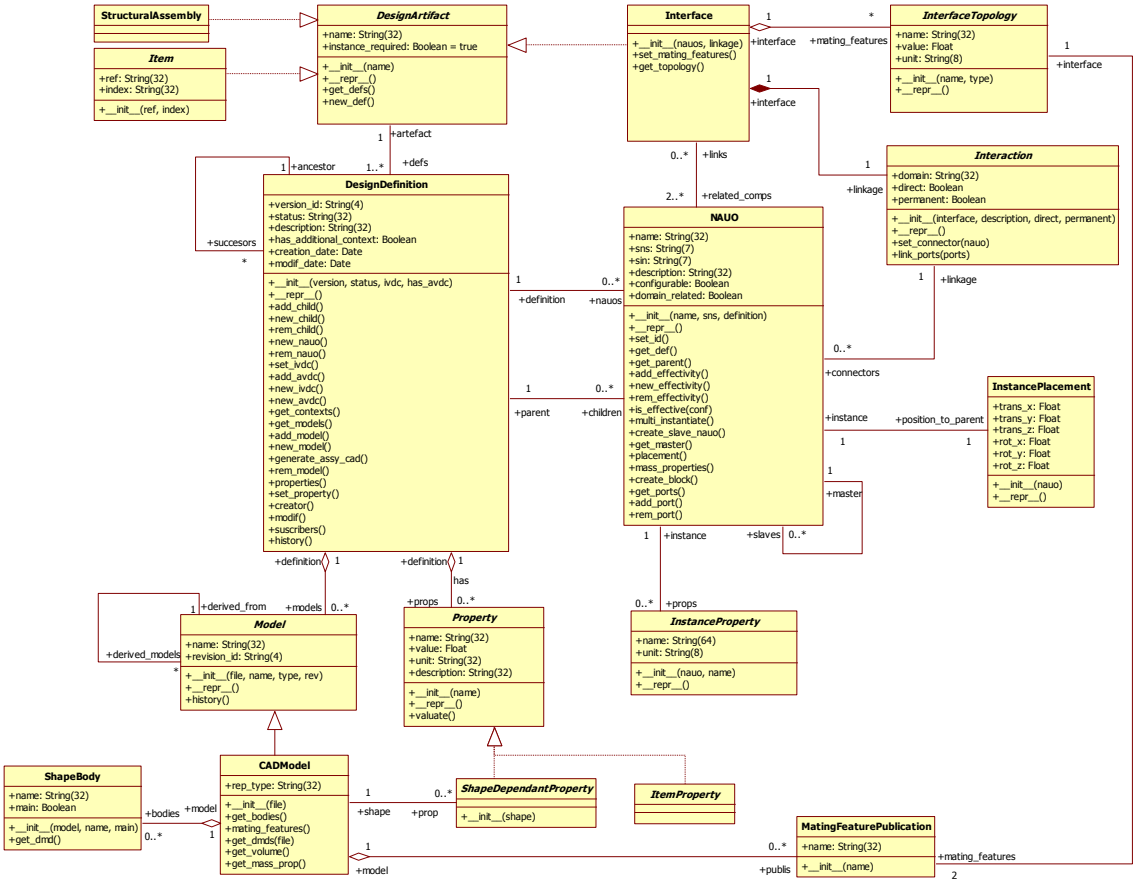


Figure 174: implemented logical data model - system artefact definition basis

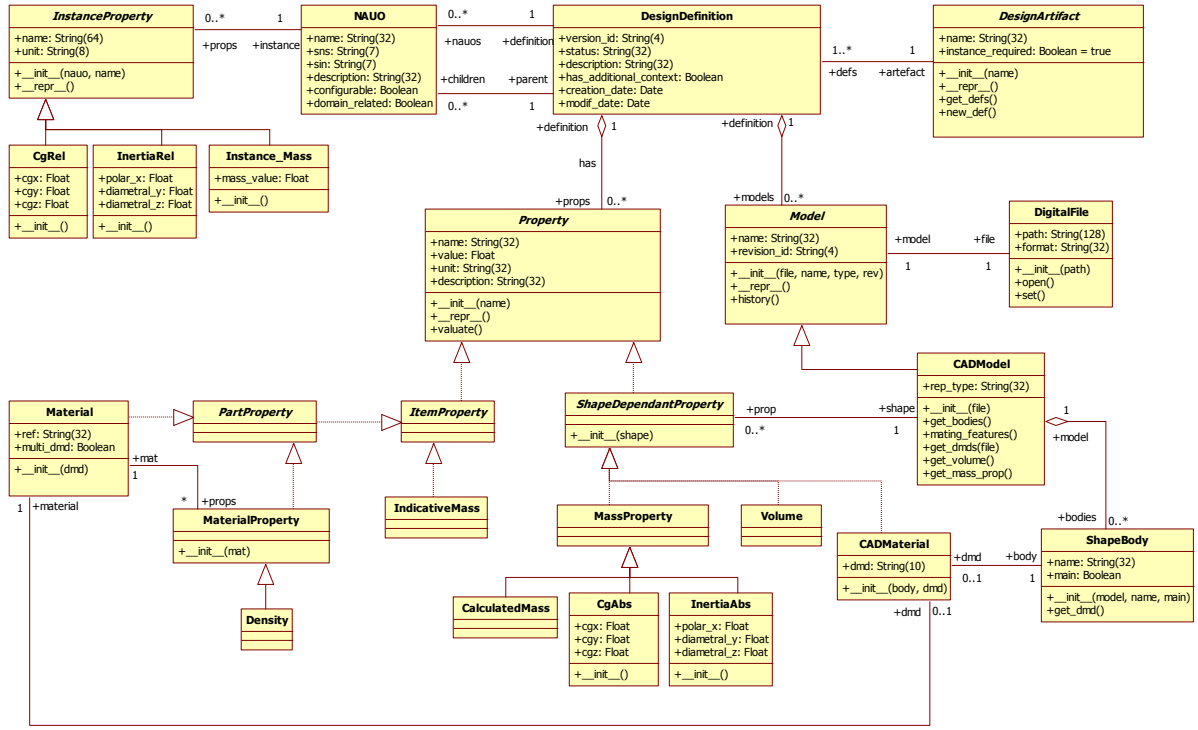


Figure 175: implemented logical data model - system artefacts and components properties

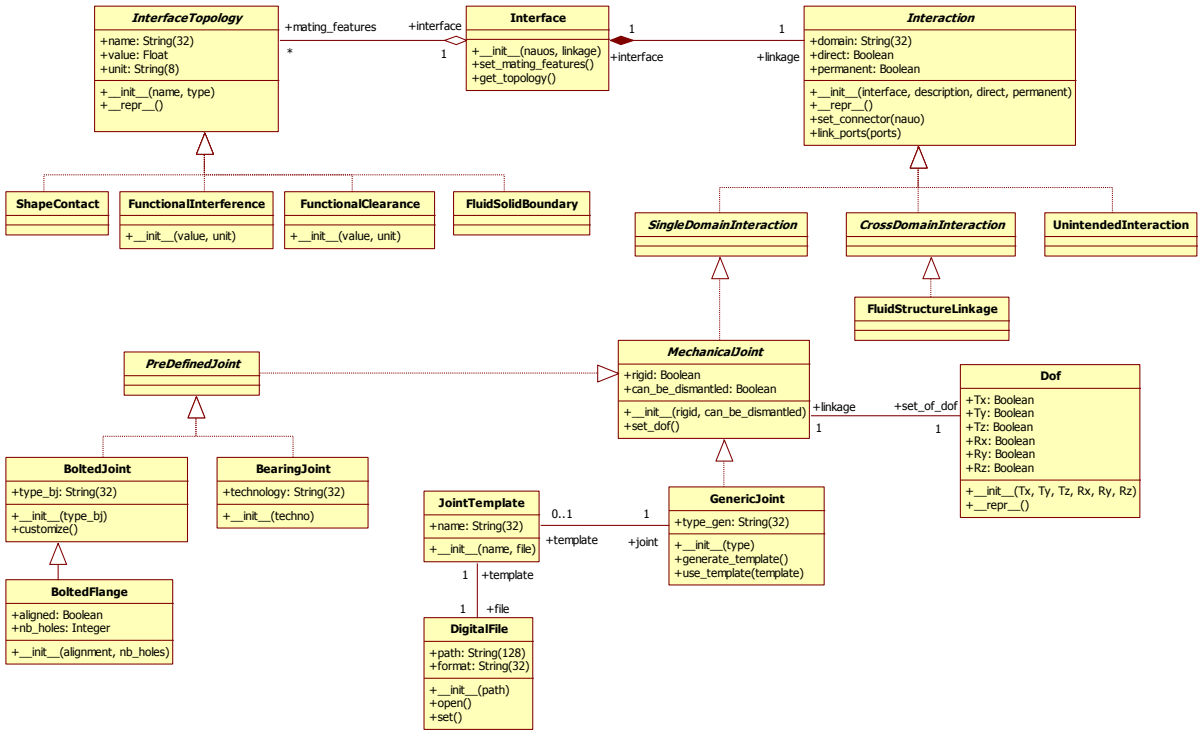


Figure 176: implemented data model - interface and interaction topology

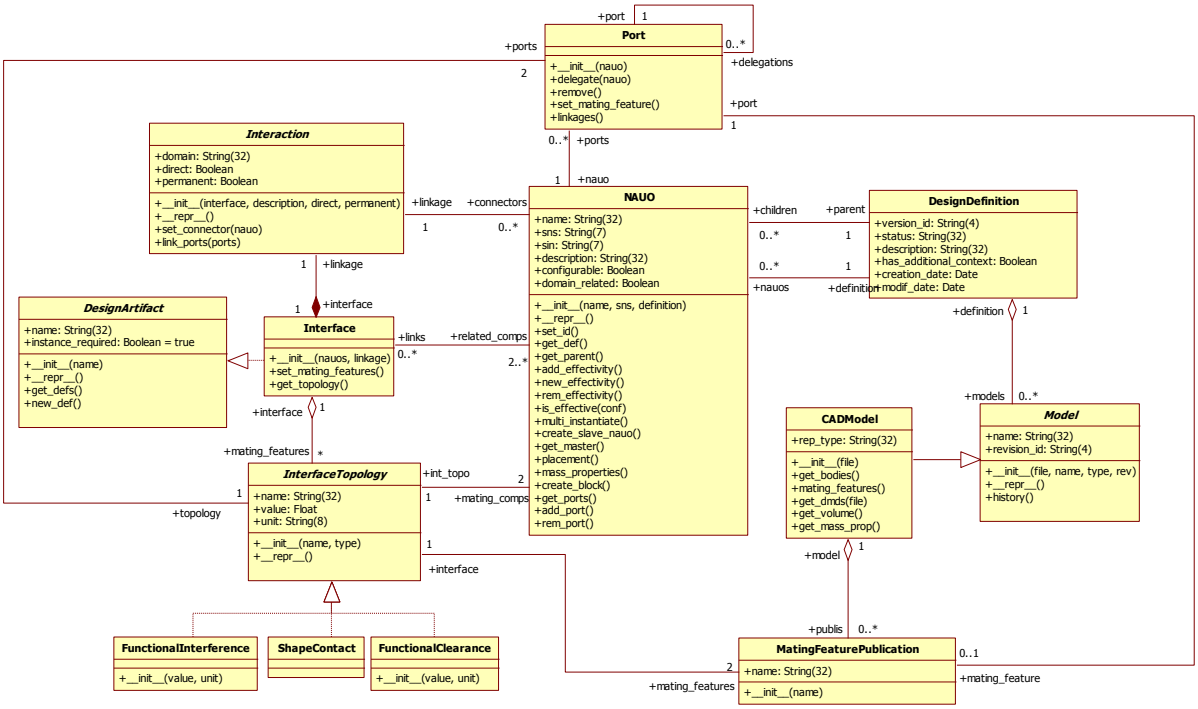


Figure 177: implemented data model - Interface and interaction definition

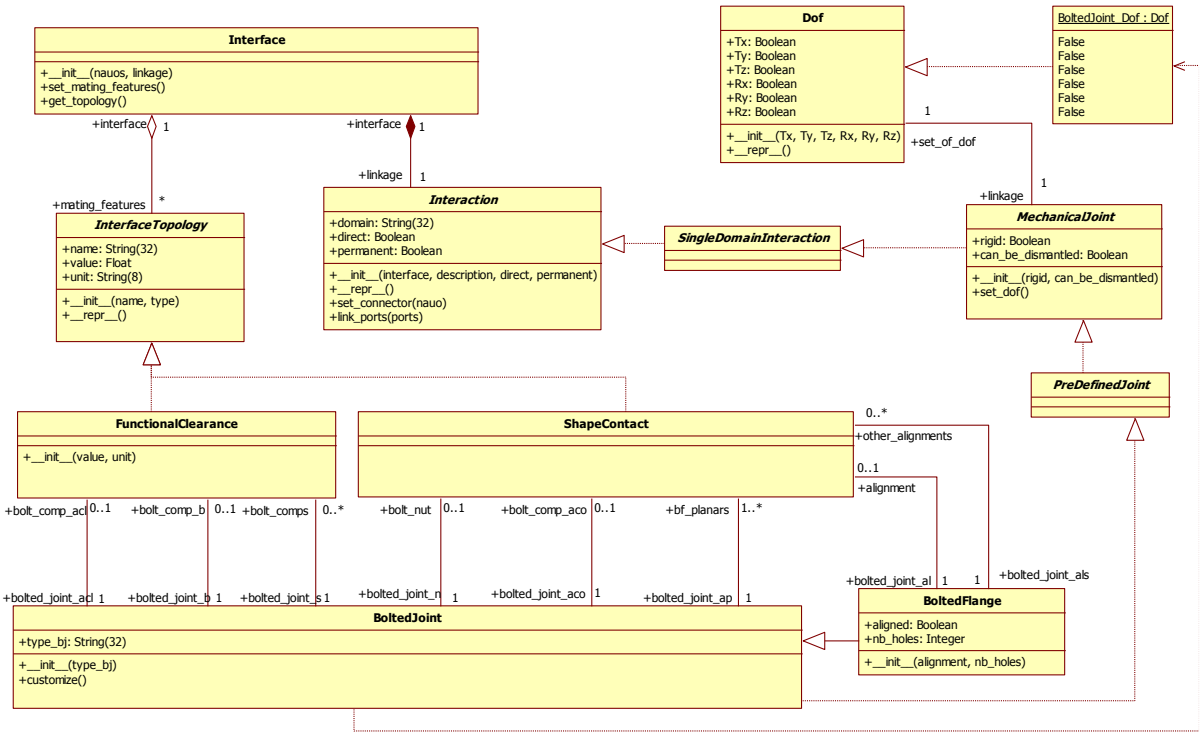


Figure 178: implemented data model - Bolted joint definition

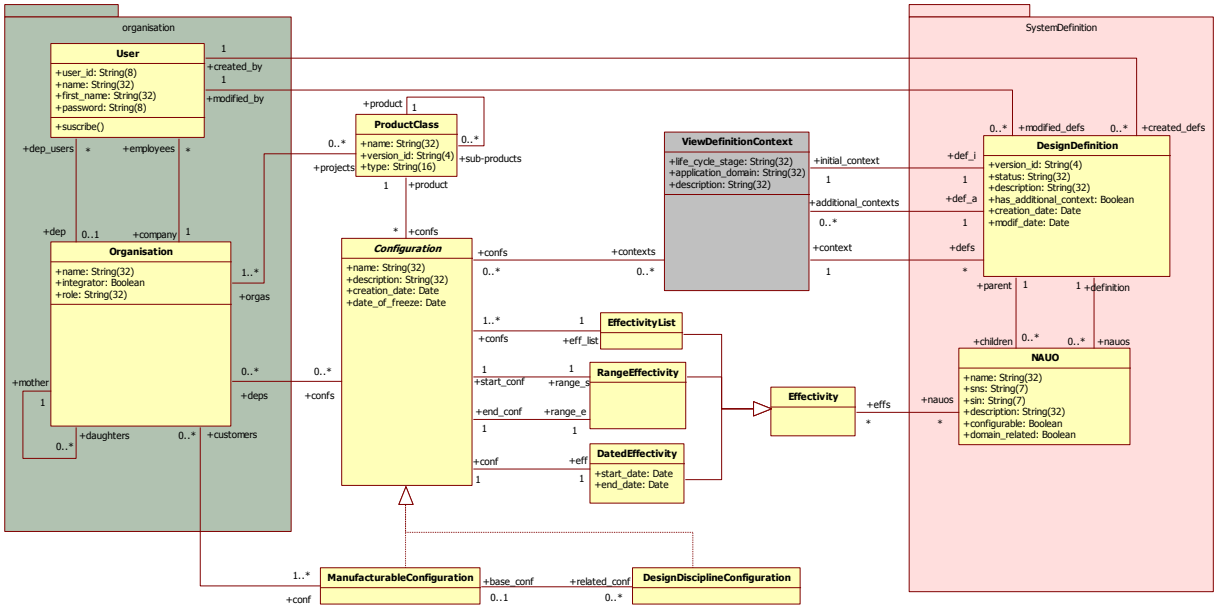


Figure 179: implemented data model - product configuration management

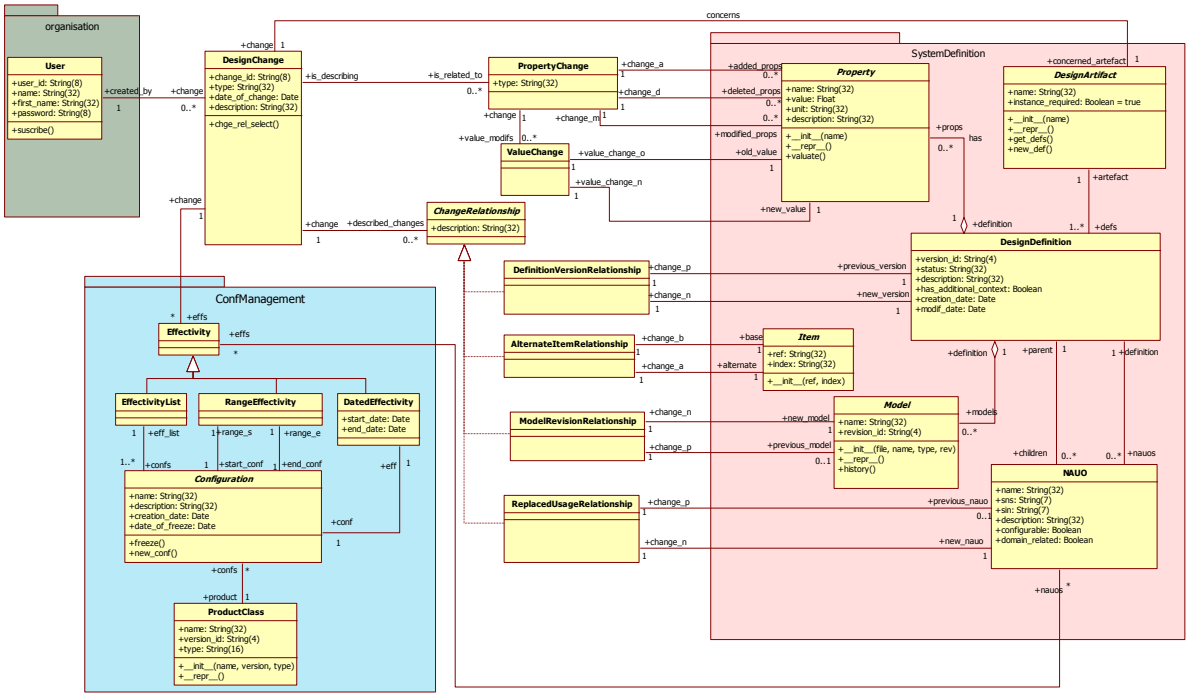


Figure 180: implemented data model - engineering change management

12.3 Generation and enrichment of the DASIF product data base

12.3.1 Generation of the DASIF relational data base system

12.3.1.1 Technological choices

In order to implement faster the logical data model introduced in section 12.2, and hence to generate the DASIF data base, we decided to use a model-driven approach to automate the generation of Python classes and related CRUD (Create, Read, Update and Delete) functions.

Python[3]	Python is an object-oriented programming language that can be used in numerous software development contexts. Python is open source and can adapt to many types of use thanks to the availability of many specialized libraries. It is particularly used by the scientific community in order to develop prototypes, and has numerous extensions for digital applications. The main use of Python in our PhD is to manage a large quantity of product data. However traditional data base tools only permit to manage relational data base systems. The complexity of the data to be managed drove us to choose an object-oriented language in order to better structure these data and be able to link the processing logic directly to the data. It is thus possible to define individually the data objects and the operations relating thereto. This has been considered necessary, but is not naturally usable in a storage system within a database.	
SQL Alchemy[4]	It was hence necessary to use a tool enabling the communication between data bases systems and data as "objects" so that they can be stored and manage in a data base. We have decided to use the SQLAlchemy library designed and developed specifically for Python. This library enables on one hand the communication with relational data base systems, and on the other hand to adapt the data structure so that they can be used in a data base thanks to its ORM module (Oriented Relational Mapper). SQLAlchemy is not the only one Python library enabling to do this, but it has the reputation to be complete and regularly maintained.	
Elixir	The syntax used to adapt the traditional python script in a python script usable for generate and manage a database being complex, we decided to use the Elixir library. This library, also designed and developed for Python, enable to drastically reduce the script length, the time to treat it, so that it can be used in a data base.	
PyQt[5]	The user interfaces of this prototype have been designed with the Qt framework for python called PySide. This framework enables to define easily and quickly portable graphical user interfaces.	

Table 7: technological choices for DASIF data base generation and management

12.3.1.2 Genepy

Thanks to a trainee work, we developed a tool enabling to automatically generate the data base script in python from a UML model created in StarUML[6]. The development of this code generator was a time-consuming task for the trainee. However, considering the complexity of the data model to implement (see section 12.2), it has permitted to considerably accelerate the implementation of this data model for the demonstrator (no need to encode the data base manually at each iteration/modification of the data model).

The generator was developed in order to generate the code from XMI (XML Metadata Interchange) files representing the UML model. We initially wanted to develop a generator working for all

[3] Official website of Python language and community: https://www.python.org/

[4] Official website of SQLAlchemy library: http://www.sqlalchemy.org/

[5] Official website of PySide: http://qt-project.org/wiki/PySide/

[6] Official website of StarUML: http://staruml.io/

XMI exports from any kind of UML modelling tool but the structure of these XMI files is specific to each tool. As a consequence and unfortunately, this generator only works for XMI exports from StarUML.

As shown on Figure 181, from StarUML, we generate a XMI export of the UML model introduced in section 12.2. Genepy takes as input this XMI file and works as a parser reading this XMI file and generating the corresponding python script enabling to generate the DASIF data base.

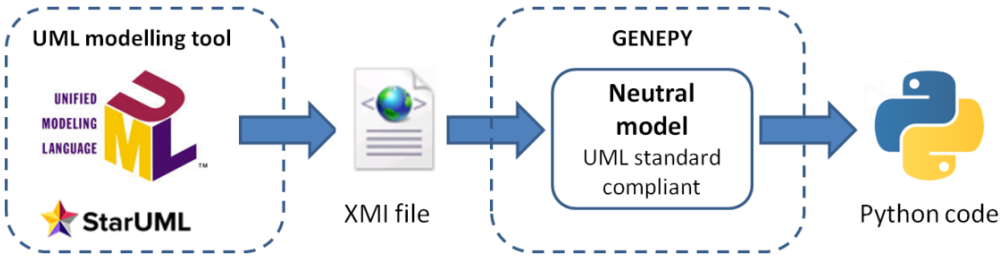


Figure 181: Working process of the GENEPY generator

The first step, targeting to translate the XMI file into a neutral model, consists in parsing the source file to retrieve the data as a tree structure. Once this structure created, each data set is analyzed and its equivalent in the neutral model is created. The second step is to sort the elements of the created neutral model in dependency order and create “strings” in Python syntax constituting the code definition of the data model. The generated code is then written into files organized into packages respecting the structure of the source data model. A Python additional file is also created to provide simplified data base connection features. Appendix X provides:

- an extract of the XMI file of the logical DASIF prototype data model (see Table 18);
- a simplified view of the GENEPY neutral data model (Figure 251);
- the generated Python source code of the DASIF data base (see Table 19).

12.3.2 Generation of the test-case DMU XML file for database enrichment

To enrich the database with test case data set, we developed a CATIA macro (VBA script) enabling to generate a XML file describing:

- the whole DMU product structure and for each of its constituting components:
  - the assembly and parts general attributes (name, functional item relationship, author, date of creation, representation_id, etc.);
  - the component position matrix;
  - the components mass properties;
- the list of CAD models representing individual parts and:
  - their attributes (DMD, name, revision, path, etc.);
  - their constituting shape bodies and their attributes (DMD, volume, etc.)
- the list of mechanical linkages captured in CATIA including:
  - the set of degrees of freedom
  - the linkage parameters (depending on the linkage technology)
  - the linkage topology; i.e. the CAD mating features

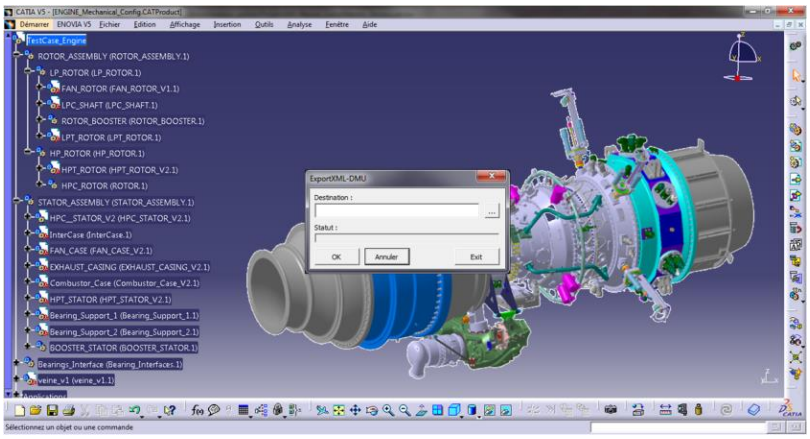


Figure 182: Launching the XML DMU EXPORT script in CATIA V5R18

The script is fully provided in appendix XI, but a simplified view of the script procedure is described in Figure 183 below.

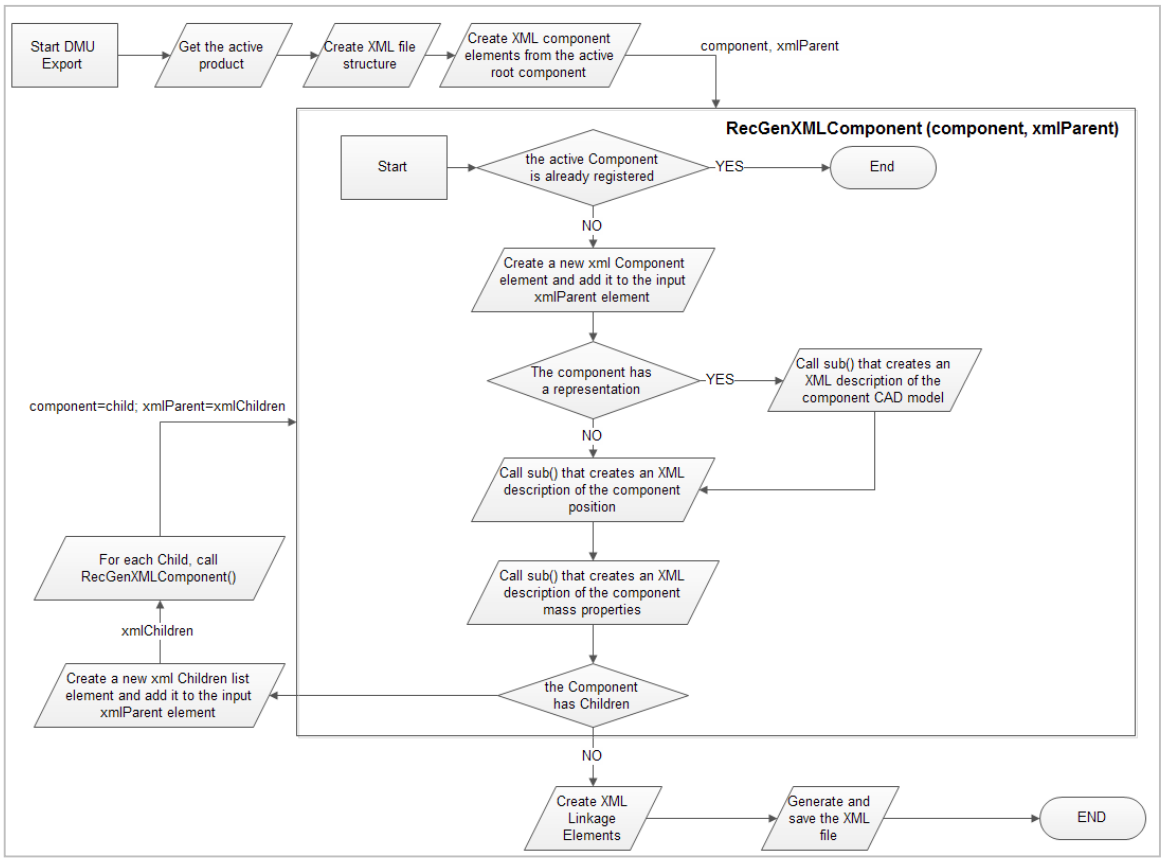


Figure 183: simplified view of the CATIA XML_DMU_EXPORT macro procedure

The script is built around a recursive function (RecGenXMLComponent) permitting to browse and describe the entire product structure and to make, for each component and for each CAD model, the necessary calculations to export positions, material and mass properties parameters. Another applicative function has been developed to create the mechanical linkages, to capture the interface CAD mating features, publish them within the CAD model and to reference these publications as the topological description of the linkages. All these information are exported in the same XML file described below:

- Figure 184 provides the high-level structure of the generated DMU XML file.

- Figure 185 shows the xml description of a component definition that includes the component attributes, its position matrix and its mass properties.
- Figure 186 shows how the product structure hierarchical links are captured within the CHILDREN_LIST elements.
- Figure 187 shows how all individual parts CAD models are captured with their attributes and their constituting shape bodies (with related material, mass and volume parameters).
- Figure 188 describes the content of a LINKAGE element including the linkage attributes, parameters and the references to the appropriate CAD mating features.

Figure 184: Structure of the DMU exported XML file

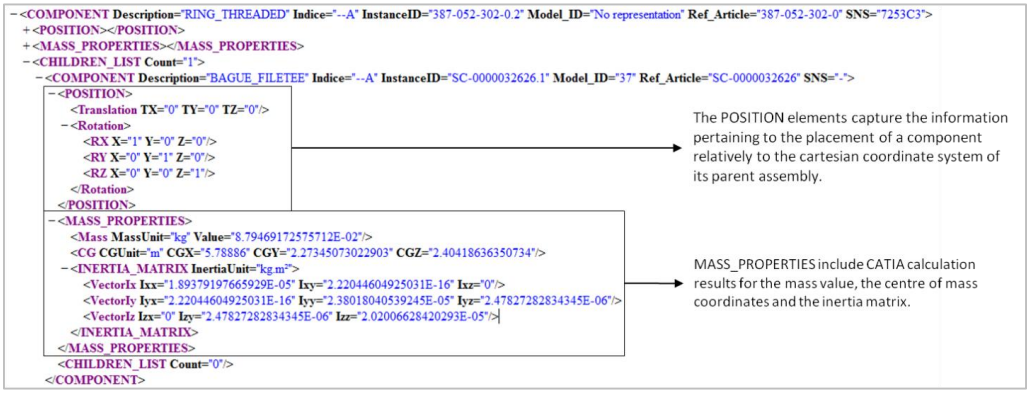


Figure 185: the xml description of a component definition

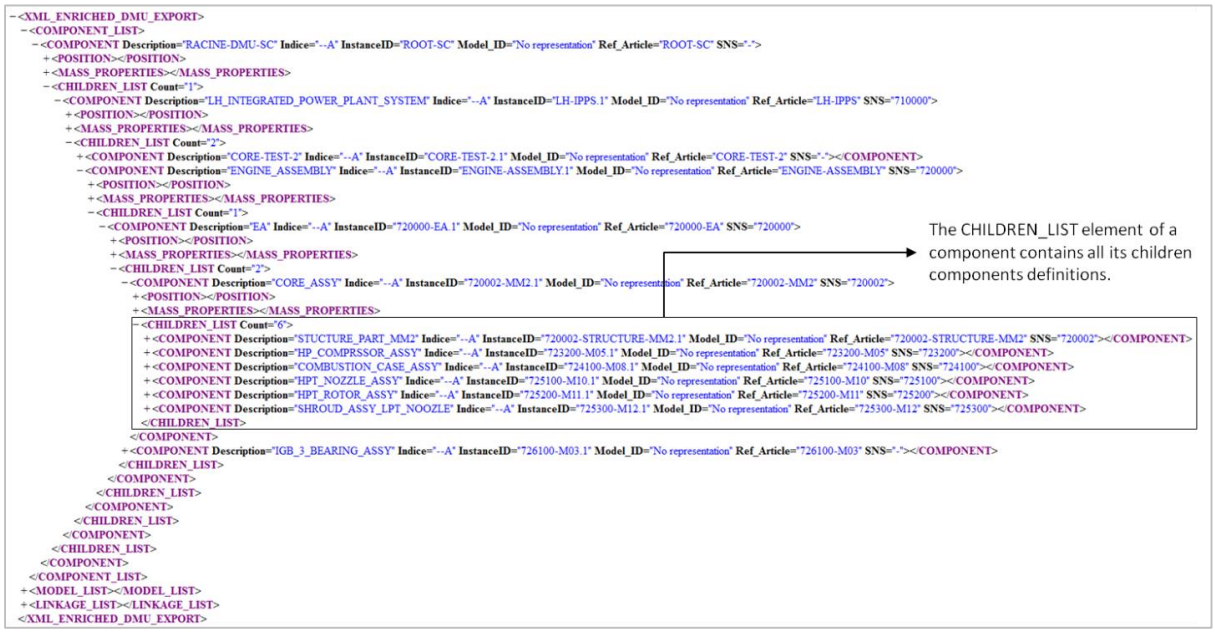


Figure 186: the CHILDREN_LIST elements recursively capturing the children components for each DMU constituent

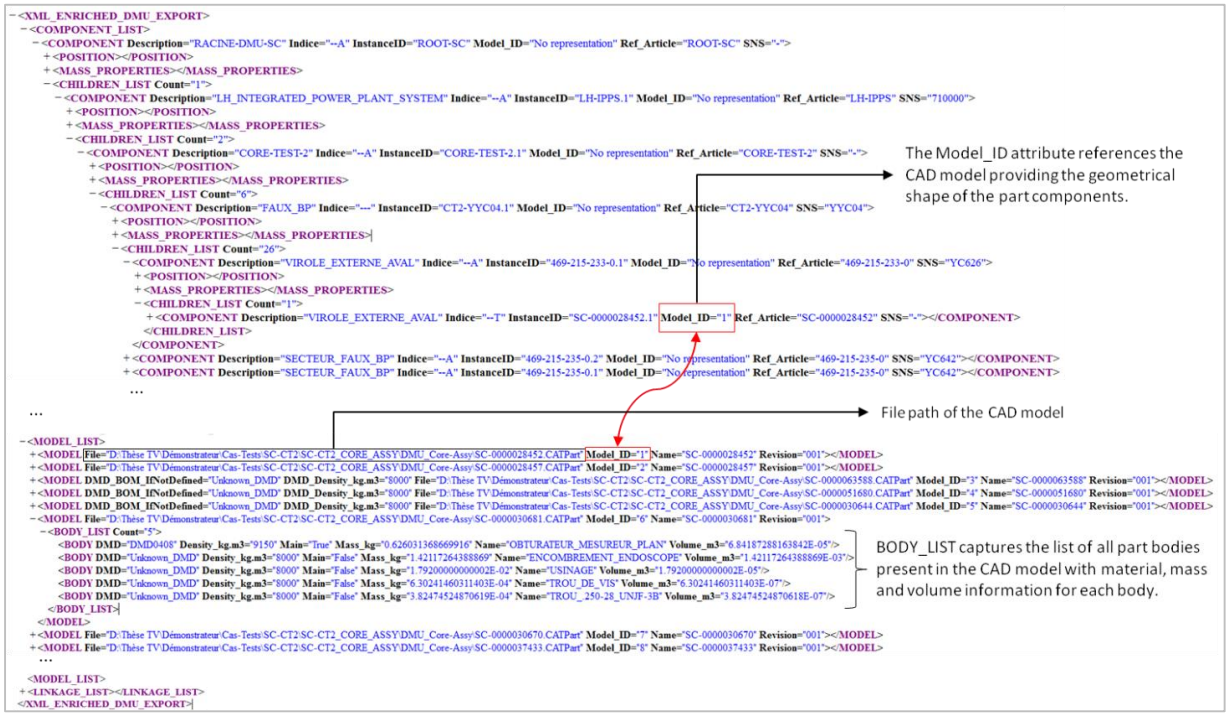


Figure 187: the Model_List capturing all individual parts CAD models, their attributes and their constituting bodies

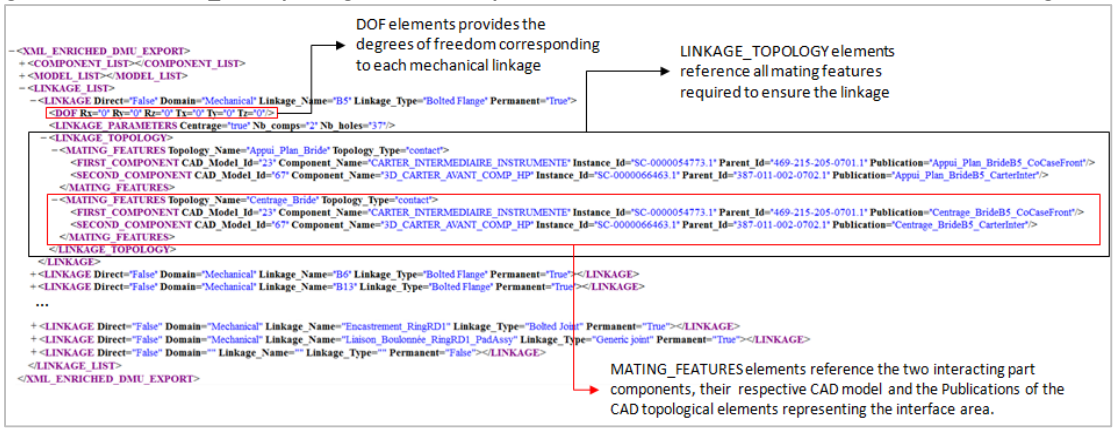


Figure 188: the LINKAGE_LIST element capturing all pre-defined mechanical linkages and related CAD mating features

12.3.3 Enrichment of the database

The python script permitting to parse the generated DMU XML file and to create the appropriate objects (DASIF classes' instances) in the DASIF prototype database is provided in appendix XII (Table 21).

12.4 Developed Functions and overview of prototyped GUIs

The generated data base code and the used SQL Alchemy and Elixir libraries enable to easily Create, Remove, Update or Delete objects in the data base as it is shown in the script enabling to enrich the database with test data set (shown in appendix XII). Now the data base server is ready to be exploited. We start to design the DASIF prototype client application (Simulation-Based DMU Manager) graphical user interfaces (GUI). These GUI have been designed with the Qt framework for python called PySide. This framework enables to define easily and quickly portable graphical user interfaces. The development of this prototype could not be finalised before the end of the PhD, but some of the main

DASIF applicative modules have been prototyped to be further developed and re-used after the PhD. Figure 189 shows the main prototyped GUI that encompass several applicative areas:
After having connected the application to the data base, the user first has to select a product in area (1). The non-configured product structure is then loaded and displayed in area (2). In area (1), the user can now request a specific ManufacturableConfiguration filtering the list by selecting a Lifecycle stage. If necessary he can also request a related business DesignDisciplineConfiguration by selecting a specific application domain. Selecting a component in the product structure, several applicative modules can be called and displayed by selecting the appropriate tab:

- Definition tab: enabling to display all the information related to the selected artefact definition (Areas (3), (4), (5) and (6) in Figure 189);
- Configuration Manager tab: enabling to create new configurations, to modify existing ones, to define artefact definition usage occurrences effectivities and enabling to compare two successive configurations and visualise and trace engineering changes between these two configurations.
- CAD tool tab: CAD viewer enabling to visualise the DMU representation of the selected component and to load it in CATIA V5 if necessary.
- System Modeler tab: corresponding to the MBSE modelling and integration framework enabling to represent the selected configuration or one of its sub-system as a system logical architecture (as explained in 10.3.4) and specify logical DDMU and BMU architectures. This module has been developed yet.
- Links graph tool tab: this module is dedicated to traceability functions. It should be a visual interface enabling to visualise all dependency links that have been established between artefact definition models or properties. This module has been developed yet.

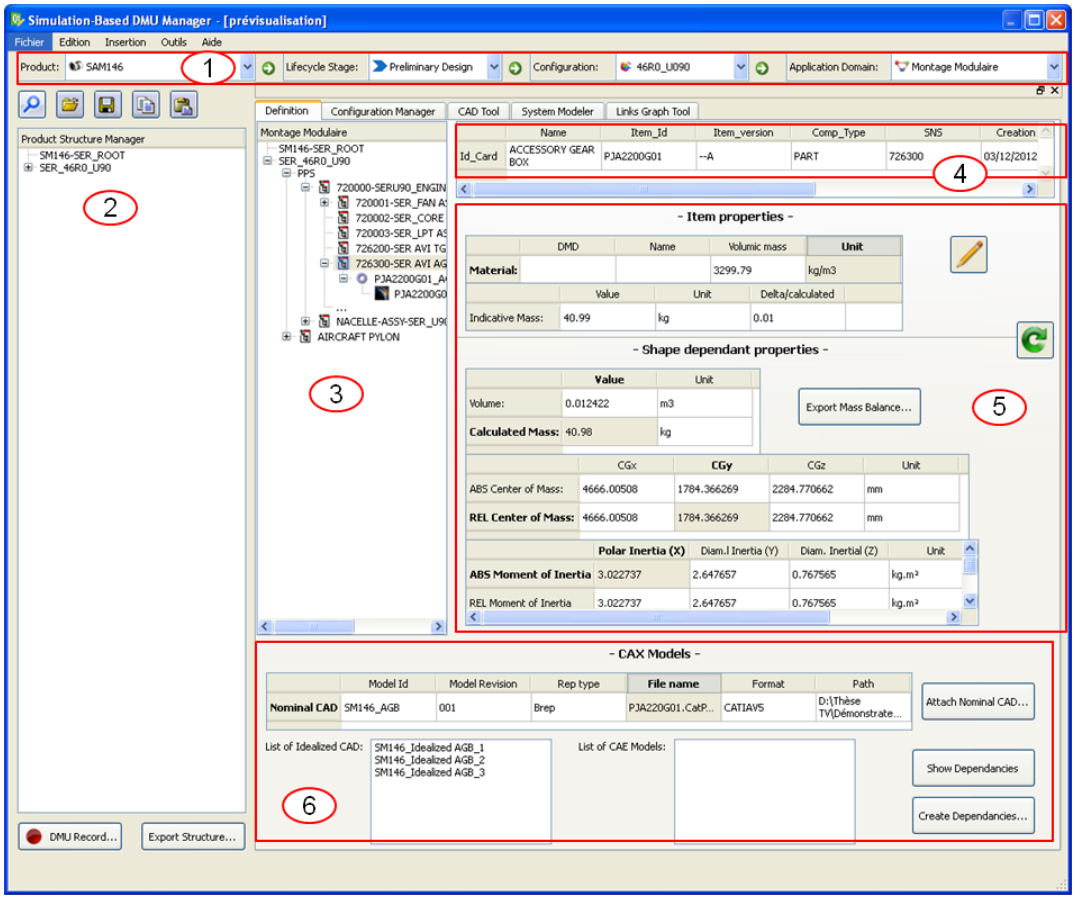


Figure 189: DASIF Prototyped GUI - Artefact Definition Form

In area (2) it should also be possible to import a product structure from the CAD system using the "DMU Record" button to directly enrich the data base without using the DMU XML export macro introduced previously. This functionality has not been developed yet.

In the definition tab, area (3) permits to browse the product structure and select a component to access to its related artefact definition. Area (4) displays the artefact definition general attributes (artefact name and identifier, artefact type, definition version, creation/modification date, etc.). Area (5) displays the properties defined for this definition; i.e. item properties and shape dependant properties calculated from the CAD model and from the capabilities offered by the used CAD system (here CATIA V5R18). If some of these properties (mass or material properties) could not be computed before, then user can modify manually the item properties and update the calculation of the inertia moments if necessary. Area (6) display the CAX models (here only CAD models) that are associated to this artefact definition. In this area, models attributes and models dependency links (versioning links, derived_from links, etc.) can be defined and displayed.

The Configuration Manager tab, shown on Figure 190 and Figure 191, encompass two sub-tabs:

- The Business Structure Manager tab (Figure 190): enabling to create new configurations, to modify existing ones, to define artefact definition usage occurrences effectivities. This module enables to define other specific business configurations (right side) and to define new component usage occurrences (NAUO) and associated effectivity (for the created slave structure) of the components existing in the master configuration (left side).
- The Structure Compare tab (Figure 191): enabling to compare two successive configurations and visualise and trace engineering changes (model changes, properties changes, meta-data changes, new created artefacts or new NAUOs) between these two configurations.

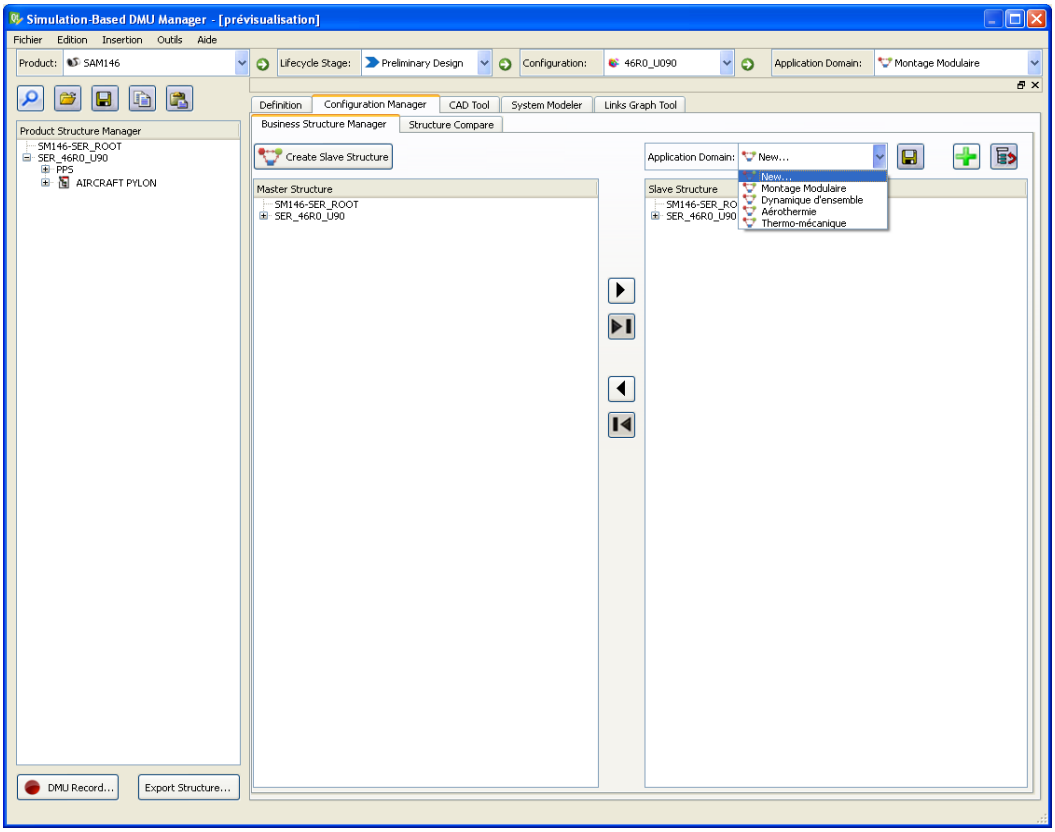


Figure 190: DASIF Prototyped GUI – Configuration Manager Module – Business Structure Manager

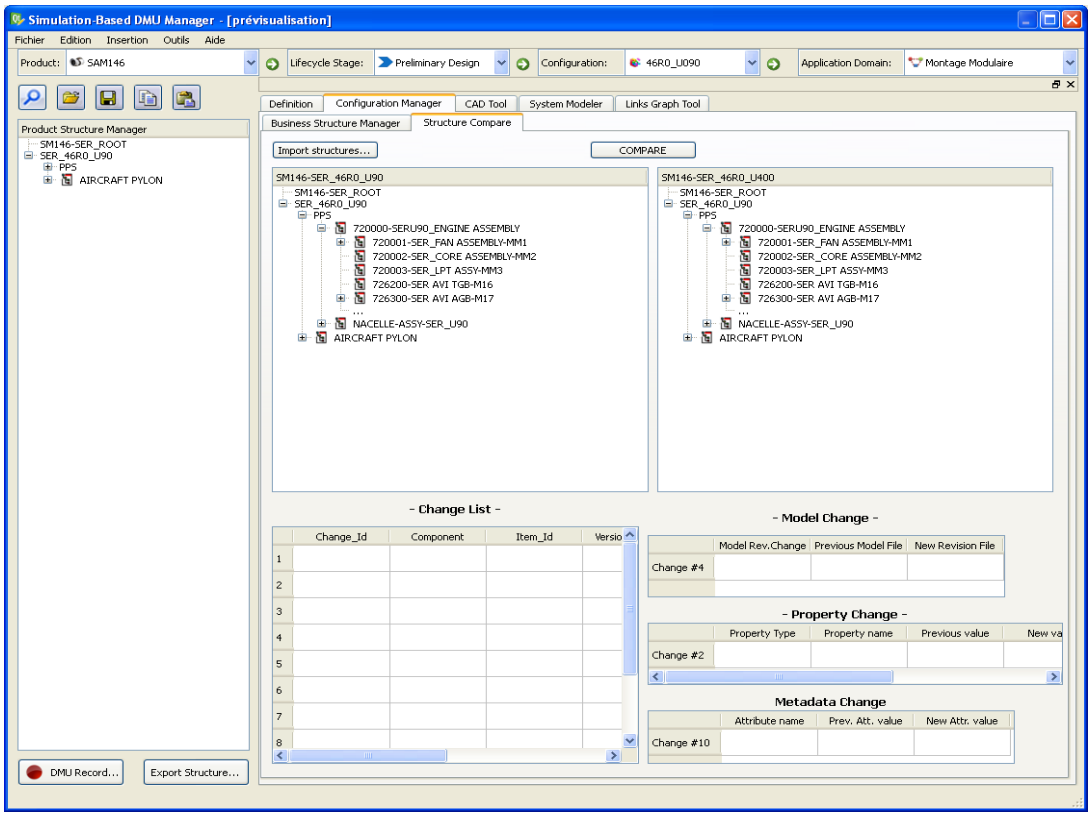


Figure 191: DASIF Prototyped GUI – Configuration Manager Module – Structure Compare

12.5 Conclusions

Most of DASIF specified capabilities, introduced in section 10.3, was still not operational at the end of the PhD. The completed data base code implementing the logical data model introduced in section 12.2 has been generated. Query and CRUD functions are already operational but are called for the moment via python command lines. CATIA API were used enabling the client application to call CATIA functions for making mass properties calculation or to load a DMU from the product configurations stored in the data base. This prototype has enabled to assess and validate the "implementatbility" of the logical data model of section 12.2. It might serve as a basis for further developments within Snecma units.

The capabilities that could not be developed have been specified in the frame of the Crescendo project. In this project, and based on the interviews and audit performed within Snecma design departments, we develop a demonstrator scenario called "Product Integration" and introduced in next chapter. Remaining DASIF advanced capabilities were hence fully specified to PLM/SLM software vendors in order to demonstrate the feasibility and "implementability" of DASIF proof of concepts in commercial PLM solutions.

The scenario, the demonstrator and related results are detailed in Chapter 13.

Chapter 13: Potential implementations in commercial tools for operational use in multi-partner projects

13.1 The Product Integration scenario and methodology

13.1.1 Introduction to Power Plant Integration use case

The product integration scenario is part of a bigger industrial use case developed in the frame of the Crescendo project: the Power Plant Integration (PPI) use case. This industrial use case implies the participation of a large supply chain, covering the airframe (aircraft or helicopter), engine, nacelle or cowlings, and component manufacturers. The engine and the airframe influence each other and drive overall product performance. The integration of the power plant in the overall aircraft architecture is studied from the very beginning of preliminary design phase, and its iterative optimisation is achieved during the detailed definition and development phases.

Nowadays, the Power Plant Integration activities are facing growing complexity in the way the product is defined and new challenges regarding engineering, business and collaboration. First, there is the complexity of the product itself. There is also the complexity regarding business with the increasing need for multi-physics simulations to better understand product behaviour as well as multi-disciplinary analyses and optimisation to improve the product definition and performance. And finally, a complexity of collaboration involving several layers of partners and sub-contractors and that requires tackling Intellectual Property Right (IPR) preservation of companies as well as managing information integration between heterogeneous information systems. The new industrial performance regarding Power Plant Integration activities (all along the lifecycle of the product) will then come from the consideration and conjunction of all of these aspects (engineering tools and methods, business and collaboration).

The Power Plant Integration challenges have been addressed by 9 Test Cases and related scenarios that consider the following issues:

- How to manage the preliminary design phases considering the process from “client expectations and requirements” to the design evaluation.
- How power plant system design optimization can be made more robust by including the subsystem and component properties.
- How the integration of subsystem models in the overall system modelling (a necessity for cross system-level optimization) will be enabled by common interface architecture. This will allow more accurate modelling of the overall system so that trade-offs between design, manufacturing and services can be more easily and accurately studied from a cross-company perspective. This “new” consistency (sustained by increased maturity) of the overall system modelling enables a better exploration of how subsystem properties affect the overall system performance for the purposes of optimisation convergence.
- How increased communication and collaboration between the system level and the subsystem level, and homogeneous modelling of the power plant elements, will reduce the amount of re-work and development lead time by a concurrent approach ensuring up-to-date configuration and model management.
- How the BDA system (see sections 3.4, 7.3 and 10.3.6) will enable a joint multidisciplinary and integrated development plan; trade-off studies for power plant optimisation; mature design data delivery for product design reviews and critical design reviews; efficient “verification by

analysis" of power plant specifications, and the development of meaningful product and design process quality indicators.

- How reduction in physical testing can be achieved, due to reduction of iteration or rework loops and higher quality, more consistent modelling of the physical properties, allowing shorter development tests (fewer configurations and better simulation quality).

The Product Integration scenario is a part of this global industrial PPI use case and addresses the following technical challenges:

- Improvement of lead time for setting-up and simulating the integrated product by managing multi-level and multi-physics analyses of complex products thanks to:
  - A systemic approach based data management for complex product to manage physical links properties and facilitate assemblies;
  - Collaborative exchanges and consistent integration of partners' technical data using the BDA object model.
- Improvement of the traceability of models and simulations, thanks to :
  - A systemic approach based data management that helps to manage links and associated contexts between the "sub-systems" of the product;
  - The introduction of the "simulation intent" that should manage the knowledge and capture of the information during the simulation process.
- Decrease of rework time, thanks to:
  - Exchange of specifications for technical data reducing interpretation issues;
  - New automated quality check processes.

Regarding the technical challenges, significant efforts have been made on the definition, specification and development of a dedicated environment for product integration called "integrator environment". This environment has been developed on the basis of a systemic approach and object-oriented system modeling formalism and language. It has been applied to the product breakdown so as to solve the issues related to CAD-CAE integration, multi-domain representation and multi-level management of the product, collaboration and interoperability. It proposes a common framework to represent and manage the product in different engineering activities during the product development process.

13.1.2 The product integration test-case scenario

The Product Integration scenario aims at setting-up an integrated mechanical analysis and ensuring partner collaboration through exchange of appropriate data sets in order to:

- Specify models interfaces and modelling properties;
- Integrate efficiently sub-systems models (FE models) coming from involved co-designers and partners;
- Feedback downstream design departments and partners with results.

The concrete objectives of this scenario are:

- Assemble automatically PPS sub-systems FEMs using the integrator dedicated environment;
- Interoperable SLM data exchange and partners collaboration via BDA BOM;
- Use simulation intents and modelling requirements to create automatically fit-for-purpose meshes;
- Quality process to assess the reliability of the models.

This scenario starts from the following hypotheses:

- The scenario takes place in detailed design phase where a consistent and exhaustive DMU is available (detailed geometries exist and are available);
- Work sharing rules already established;
- FE models to be integrated come from several partners using different tools and data formats;
- System architectures and mechanical interfaces between components are already defined.

Figure 192 below illustrates the process of the Product Integration scenario showing an integration chain between the Power Plant system (PPS) integrator (e.g. Airbus) and the engine model provider (e.g. Snecma) which is itself an integrator of the different engine's modules. The integration process scenario will be showed for the integration of the engine inside the IPPS. This demonstration scenario involves two different partners with their respective SLM environments and the BDA object model which ensures the interoperability and the information exchanges between these partners.

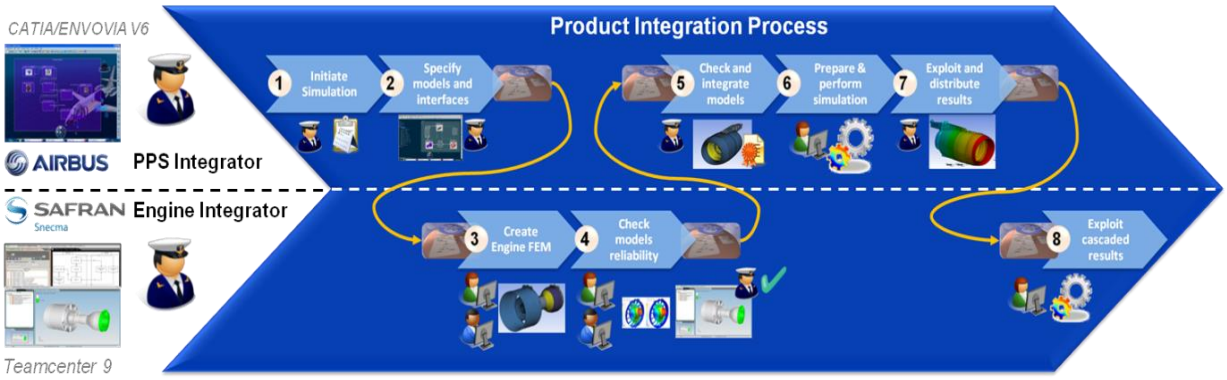


Figure 192: Process description of the Product Integration scenario

Here is the description of each scenario's steps:

- STEP 1 – Initiate the simulation: In this step, we capture the simulation context and the simulation intent so as to define the appropriate activities and integrate them within the global design process. We illustrate the re-use of pre-defined simulation processes (through the use of simulation templates).
- STEP 2 – Specify models and interfaces: The integrator collects and prepares the models for the study from the DMU, and sends all the modelling requirements needed for the engine supplier in order to obtain the required finite element model.
- STEP 2-3 (first arrow): the objective of this step is to demonstrate the potential use of a common BDA Business Object Model (BDA BOM) used for heterogeneous SLM data exchanges. The BDA BOM is the common language developed within Crescendo. In this step, BDA BOM modules have been implemented in a semantic data mapping tool enabling partners PLM systems to **understand this language by mapping their respective data models to the BDA BOM.**
- STEP 3 – Create engine FEM: in this step, we demonstrate the use of simulation intents and templates (including simulation context and associated models and interfaces modelling requirements) for performing automatic fit-for-purpose meshing.
- STEP 4 – Assess and check models quality/reliability: this step demonstrates how an automated quality check procedure can be set-up to generate models quality criteria and hence assess recued models' reliability and provide iinformation about the potential accuracy of integrated analysis' results.
- STEP 5 – Check and integrate sub-systems FE models: Similarly to export package to supplier, the integrator imports the engine model in its own PLM system through a dedicated process

template. The automated process enables to check the compliance with modelling requirements and to integrate the engine FE model within the assembly model.

- STEP 6 – Prepare the IFEM and perform simulation: The assembly model is ready to be executed, the boundary conditions are applied on interfaces and the engineer can perform the simulation.
- STEP 7 – Exploit and distribute simulation results: once the crash landing simulation performed, the post treatment can be started. If the results are satisfying, then they are distributed to sub-system models suppliers in order to perform more accurate simulations at subsystems and component levels.
- STEP 8 – Exploit cascaded results: in this step it is shown how the engine integrator (engine model provider) retrieves the interface loads derived from the previous IPPS simulation and automatically apply them to its engine FEM in order to perform a more accurate analysis at the engine level.

Appendix XIII provides a more detailed representation of the Product Integration process in BPMN[7].

13.2 Developed prototypes

13.2.1 Industrial test data set

The test data set used for the demonstration are the CAD models constituting the PPS DMU designed by Crescendo partners (see Table 8), the FE models created for the mechanical crash landing simulation of the integrated PPS (see Table 9) and the mechanical system architecture and interface specifications used for automating the assembly of these FE models (Figure 193 and Table 22 (appendix XIV)).

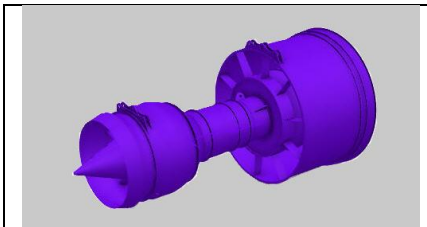
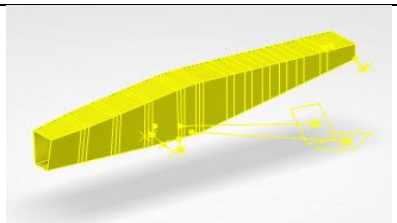
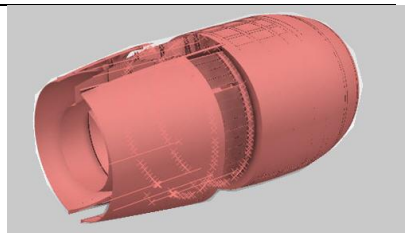
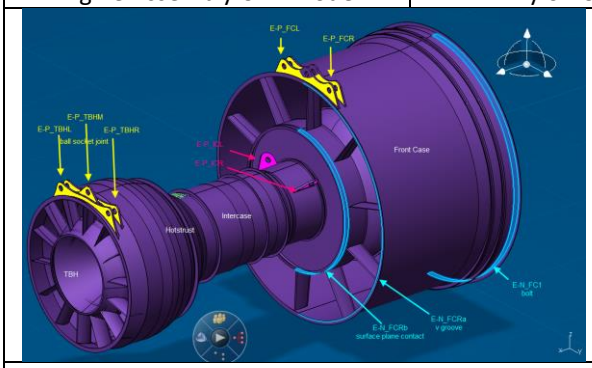
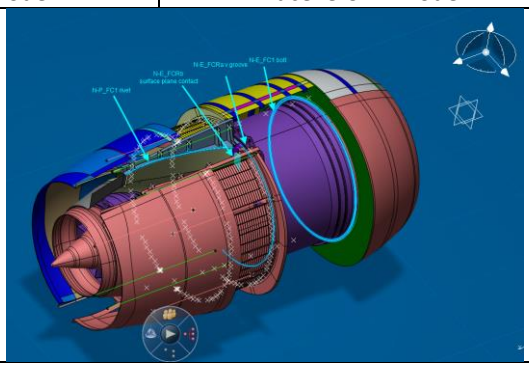
		
Engine Assembly CAD model	Pylon CAD model	Nacelle CAD model
		
Engine DMU with interface CAD mating features	IPPS assembly model with interface CAD mating features	

Table 8: DMU CAD test data set for Product Integration scenario

[7] Business Process Model and Notation : http://www.bpmn.org/

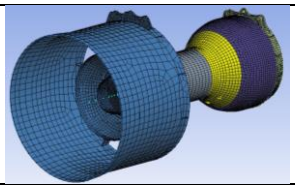
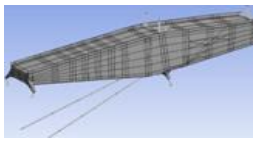
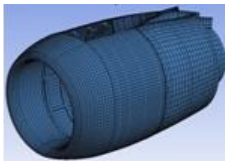
		
Engine Assembly CAD model	Pylon CAD model	Nacelle CAD model

Table 9: FE models test data set for Product Integration Scenario

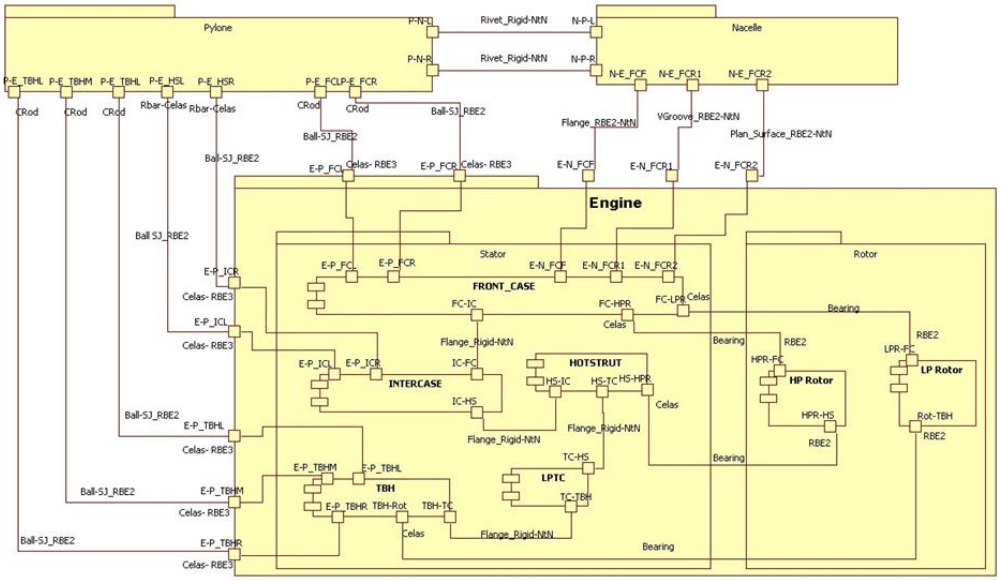


Figure 193: Mechanical system architecture and interface specifications for IFEM creation

13.2.2 Demonstrators

13.2.2.1 STEP 1: Initiate the simulation

The Integrator chief engineer (from Airbus and using CATIA/ENOVIA V6) creates a study process in order to log all information about the crash landing simulation: he uses a search function to find a standard simulation process template. He can navigate in a library of methods and select the appropriate template. He can then instantiate it. As shown in Figure 194, he populates the study template through a simple panel where he defines and populates characteristics of the study (crash landing simulation), and he references inputs like product structure root from DMU and corresponding behavioural architecture system view. The process is configured for the crash landing study. The study can now be started. The study template is composed of the following attribute groups:

- Simulation context:
  - product and configuration: selection of the product configuration to access the appropriate DMU;
  - domain of the study : mechanical/ thermal/ CFD/ Thermo-mechanical, select mechanical.
- Study features:
  - name of the study;
  - type of the study: depending of the study domain (structural analysis/ Dynamic/ NVH, linear or non linear , select linear value, etc.);
  - Behavioural scenario and/or situation of life:
    - Mission profile: take off, cruise, landing, etc.
    - Situation of life: crash landing, whirling, FAN blade-off, ice ingestion, etc.
  - solver to use.

Once the template instantiated and the inputs data retrieved, we can visualise the whole study work breakdown structure defining all sequencing activities to perform for this study (see Figure 195).

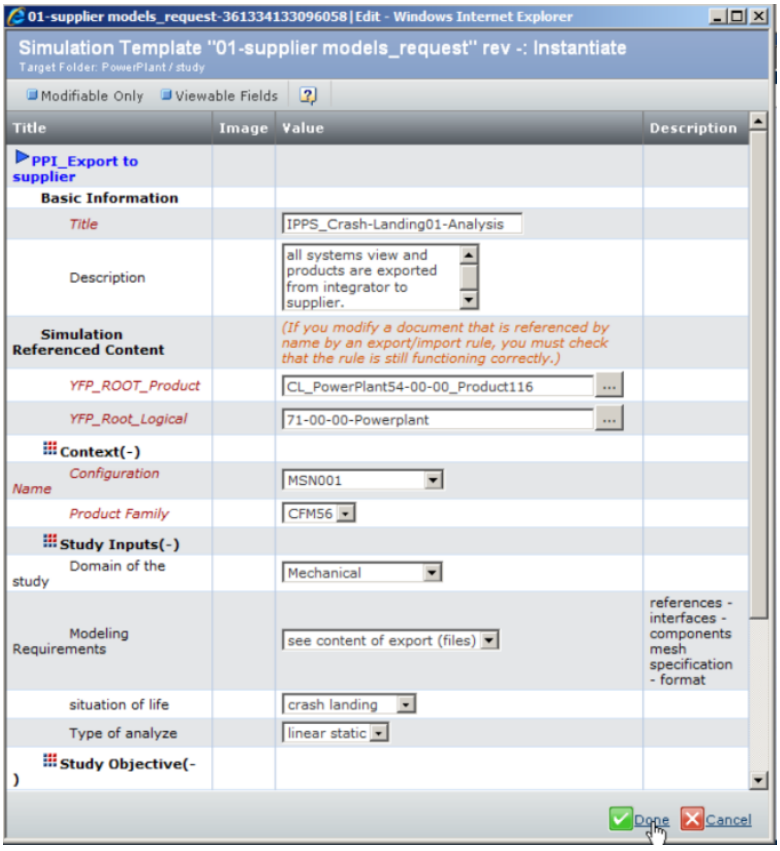


Figure 194: Re-usable simulation template used to define simulation context and intent

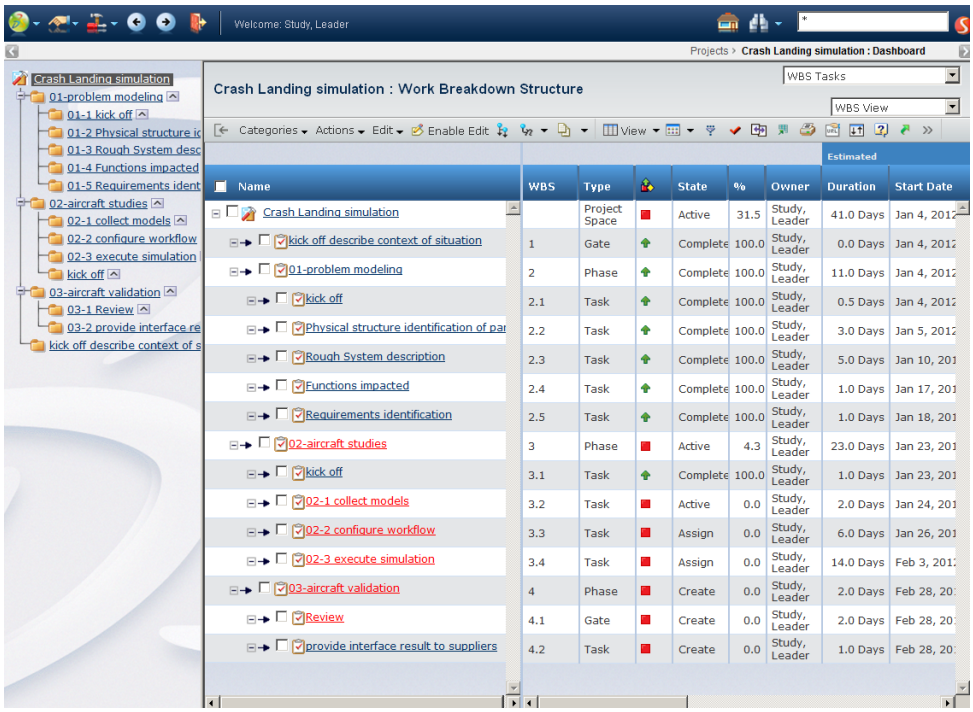


Figure 195: Retrieval of the work breakdown structure for the study

As illustrated in Figure 196, we access to the related technical data packages including inputs to use (DMU data, system view, requirements and design parameters to assess, etc.) and outputs to deliver (CAE modelling requirements, FE models, Quality reports, etc.).

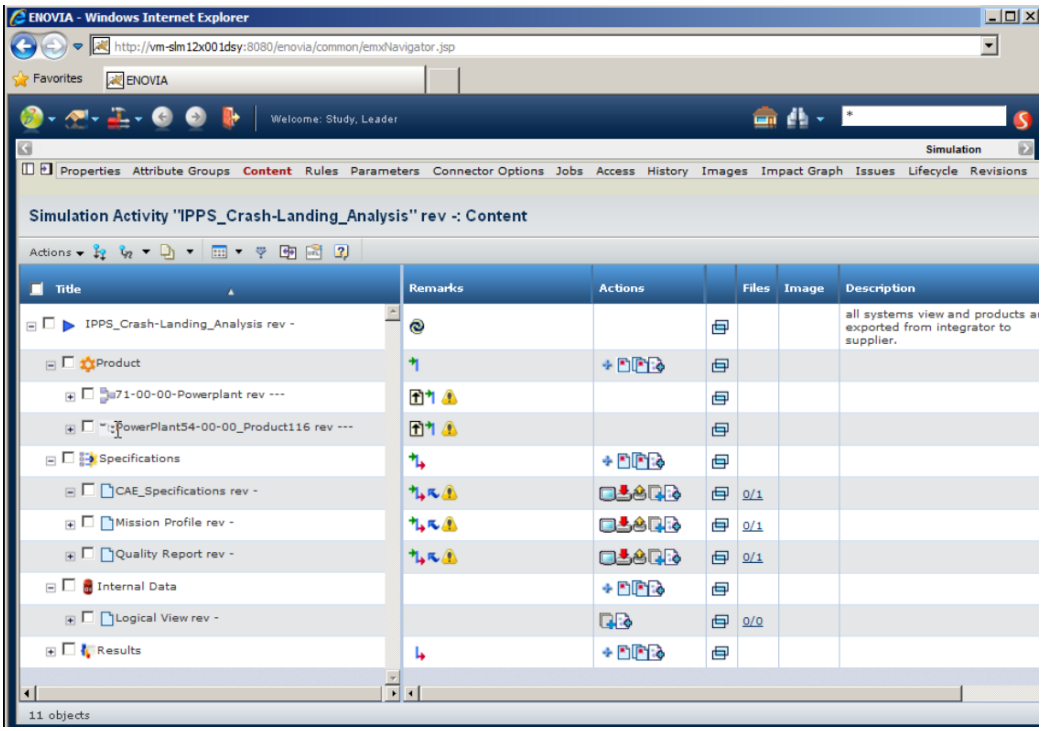


Figure 196: retrieval of the simulation workflow definition and related technical data package in ENOVIA V6

13.2.2.2 STEP 2: Specify models and interfaces

The OEM can start the study. Its teams first retrieve the root component of the DMU assembly model. As integrator, they use a specific integrator dedicated environment enabling to represent the logical architecture of the physical DMU assembly and to start specifying the behavioural architecture for the creation of the PPS IFEM. They retrieve a standard system view showing the complete assembly product (see Figure 197 below).

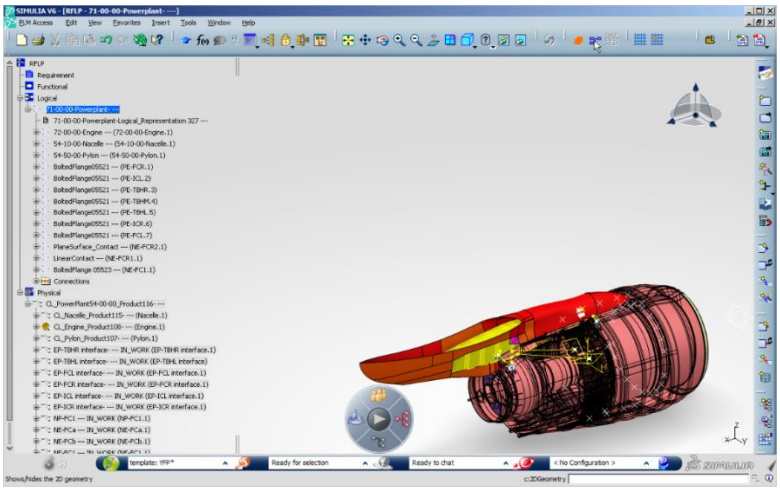


Figure 197: referential PPS DMU before transformations

However they only need the most important parts. Here engineers can filter the system root according to the discipline of the study (in this example: a mechanical study). Same thing is possible on the 3D CAD, once filters are applied both on system and 3D view, it is possible to store the filter and save it. If the analyst needs to modify or simplify the geometry, he can duplicate only the needed representations. This prevents any sensible augmentation of the data volumetry. When the prepara-

tion of the coupled models is ready, the user references the filtered models in the previous study process. So the specific study view can be directly opened. He keeps only necessary components and data that are relevant for the discipline of the study.

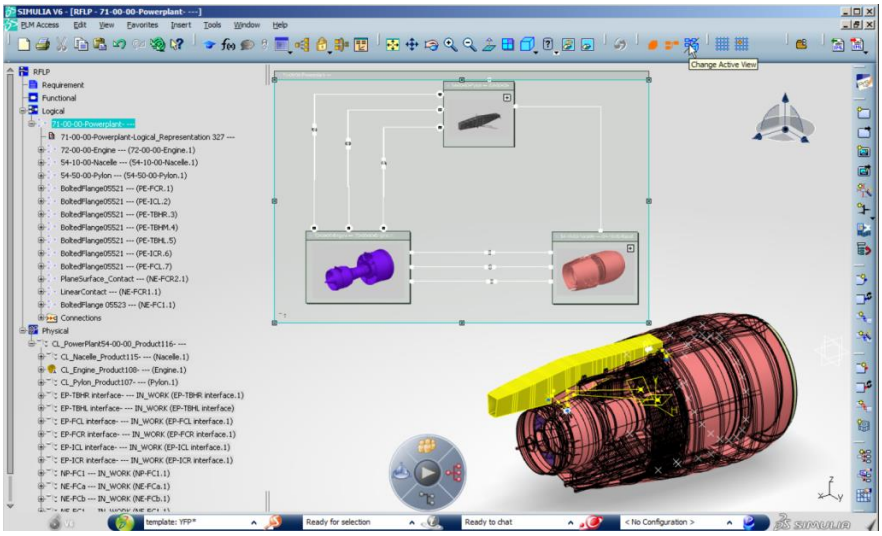


Figure 198: SIMULIA V6 RFLP framework - Logical architecture and corresponding DDMU after transformations and filters

The PPS mechanical integrator has now all necessary information to define the simulation requirements for a crash landing analysis: needed FE models and interfaces specifications.

Identify and publish the interaction areas (CAD mating features)

A Ports-Publications association referencing between the CAD mating features defined within the DMU and the system's ports is required. In order to match both systems and 3D views, the user can set-up "implement-links" between the systems and the corresponding 3D parts. Here, the engineer identifies the interfaces of the engine within the PPS DMU assembly (see Figure 199 and Figure 200). One interface has a physical (geometric) representation through publication of surfaces from the CAD models, and a logical and behavioural representation in the system view through ports.

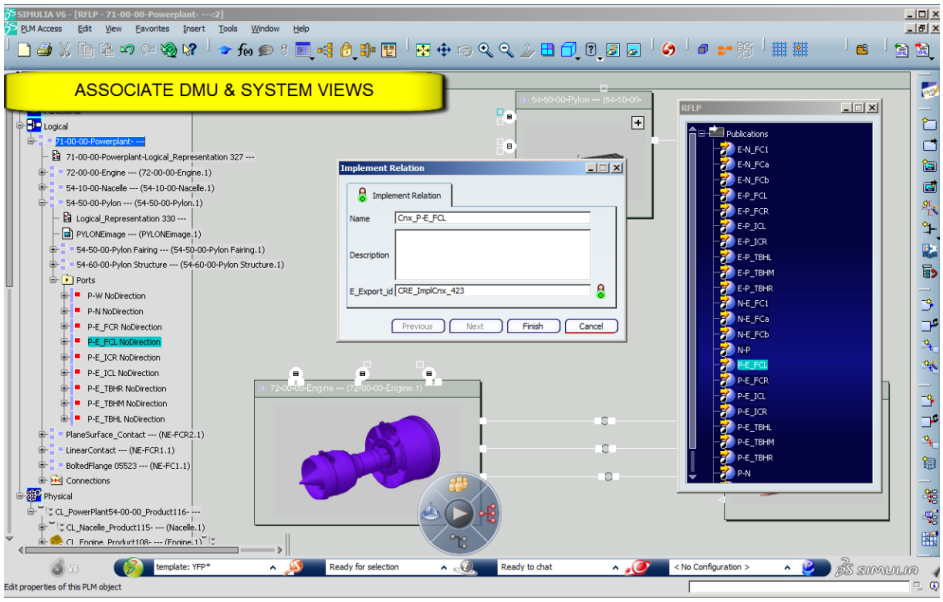


Figure 199: Referencing and Associating Ports-CAD Publications links in CATIA/SIMULIA V6

For large systems like an aircraft or an aero-engine, capturing manually all these mating features can be a really tedious and time-consuming task. That is why automated services are needed to automatically identify within a system model the geometric features of the sub-systems models that are in contact with other adjacent sub-systems models. We propose to automate this tedious task through the use of a modelling infrastructure whereby all solid and fluid domains are represented as neighbouring volumes in a subdivision of the complete space containing the model. This approach is sometimes called **cellular modelling**, and is available in some of the commercial geometric modelling libraries. With this method, the user can make use of higher level and hence more stable entities within the model, such as solid cells, to define interfaces (see Figure 139). When the user navigates on the system view, he can visualize the corresponding location of these interfaces in the geometric view. Here the association between system view and 3D view is complete.

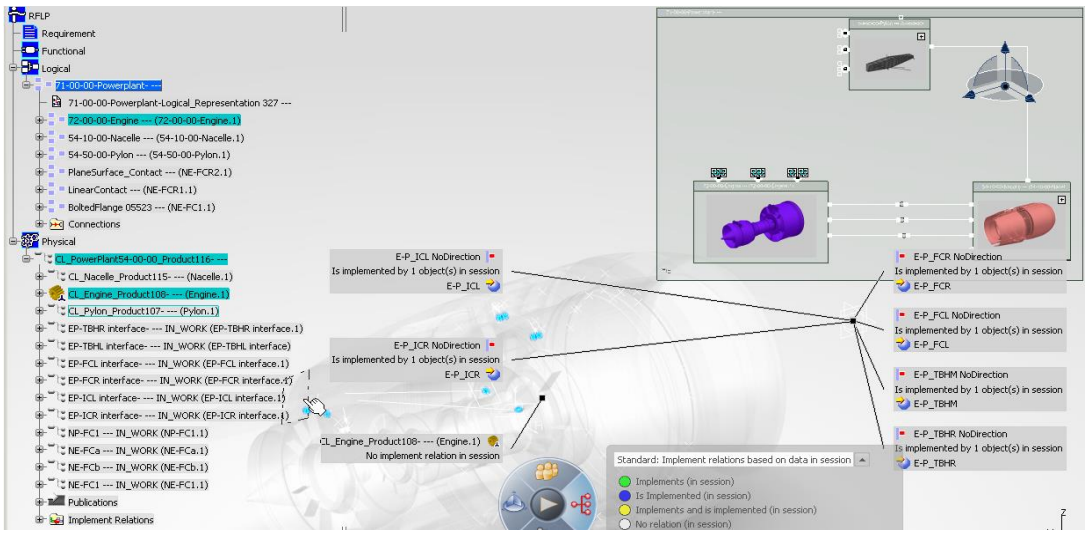


Figure 200: Visualising Ports-CAD Publications "implement links" in CATIA/SIMULIA V6

Select the appropriated technical connection from catalog for logical parts

Here we illustrate how the integrator can specify behavioural connections for the crash landing study and more precisely how he defines the appropriate connection between pylon and engine models. Figure 201 shows how the user browses an interface catalog/library which contains all standard connections.

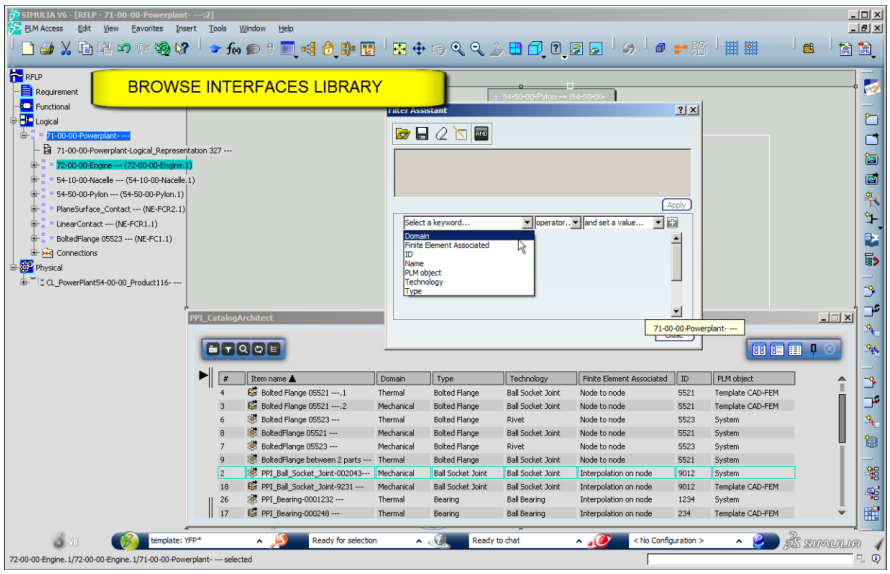


Figure 201: Interfaces catalog request engine

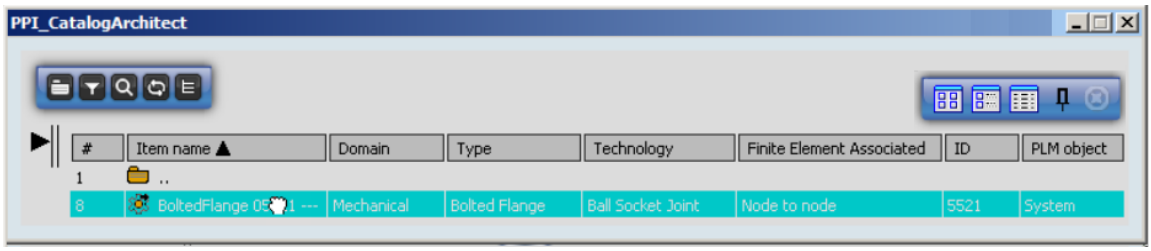


Figure 202: result of the request in the interfaces catalog

In Figure 203, the mechanical integrator instantiates 7 times a bolted flange connection.

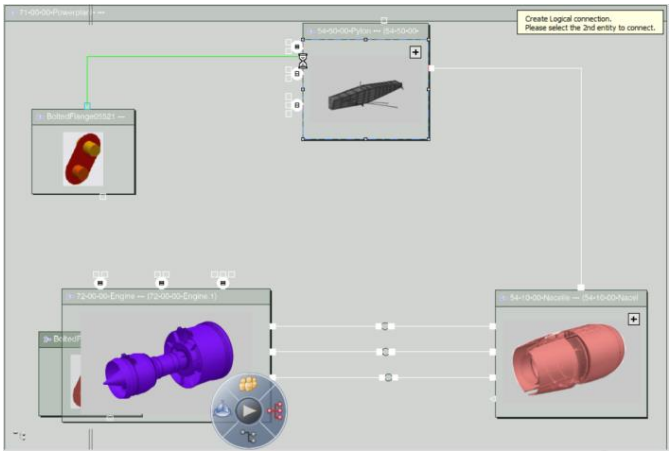


Figure 203: Instantiation of the parameterised bolted flange connection template in the logical mechanical architecture

The interface blocks parameters which are mesh specifications (size and type of mesh, maximum number of nodes, etc.) can now be instantiated directly in the RFLP environment as shown on Figure 204 below.

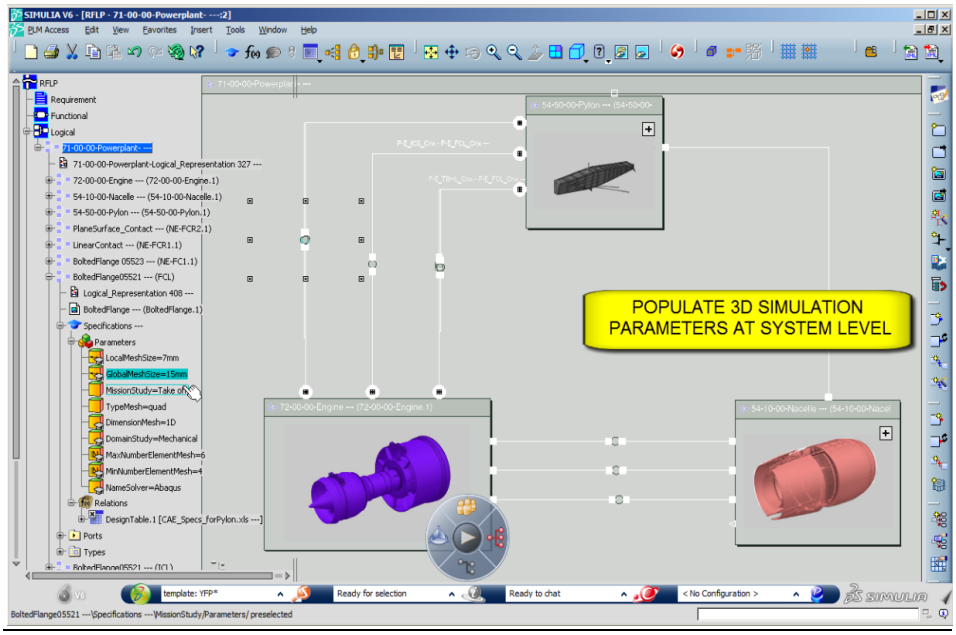


Figure 204: instantiation of interface FE modelling parameters

That way the 3D simulation intent is defined at system level. As with the system view, the engineer instantiates a physical template: he creates the bolted flange geometry and Finite Element model of

the interface from a catalog. The creation of a CAD-FEM geometric interface template is displayed in Figure 205.

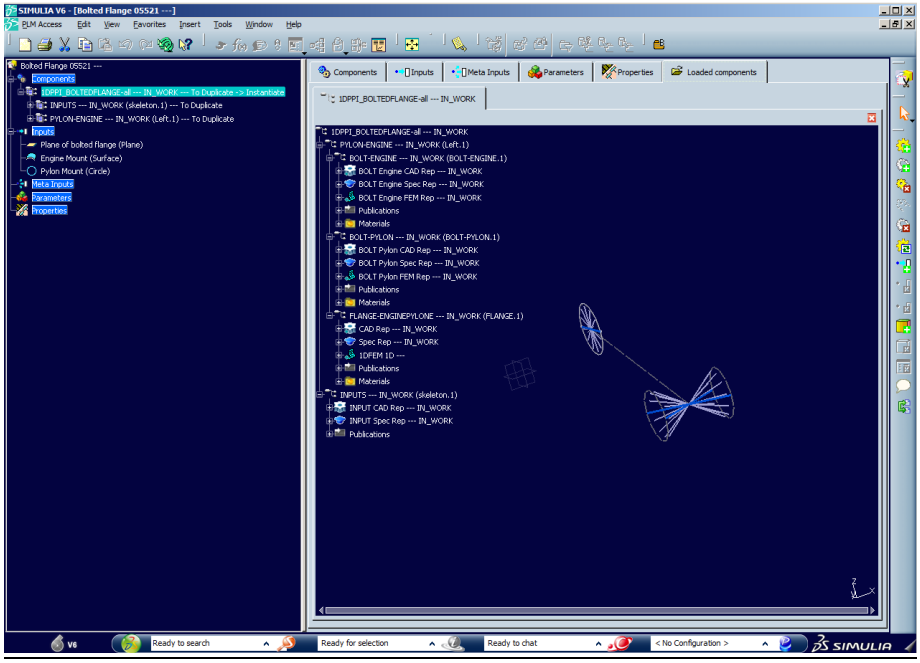


Figure 205: Creation of a CAD-FEM geometrical interface template

The filter used previously to find the interfaces in the catalog is called again in order to find and instantiate the corresponding “fit for purpose” mesh template. Then, as shown in Figure 206, the user should select the appropriate CAD interface publication where the mesh template needs to be instantiated and he chooses the orientation of the mesh representation.

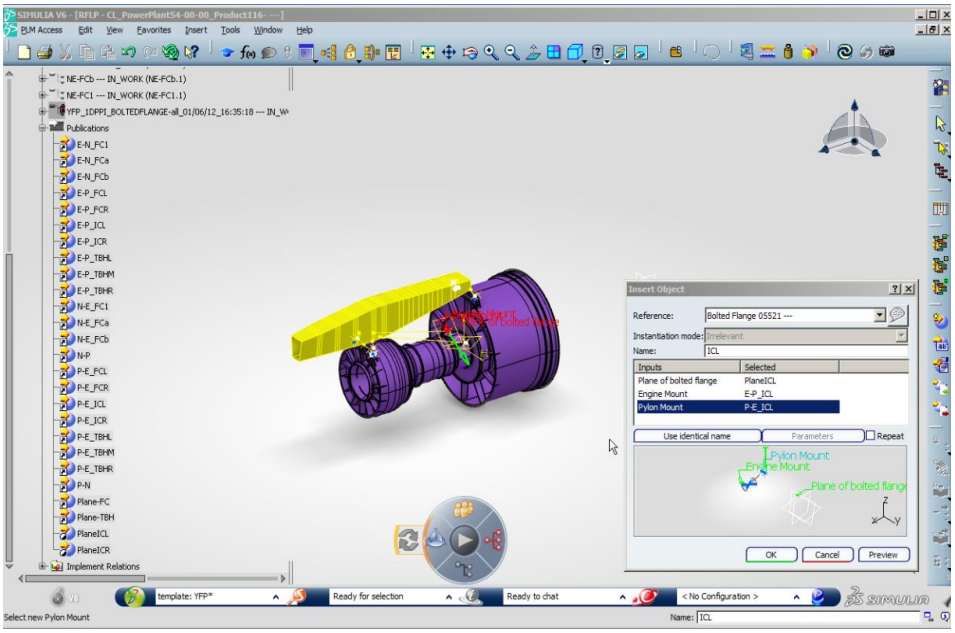


Figure 206: CAD-FEM geometrical interface template instantiation – definition of expected mating features

The implement link can be precisely traced, so that if a displacement is supposed to be applied over a port, it is possible to know exactly on which surface that displacement should be applied. This method enables precise definition of the interfaces on the FE models that shall be provided by suppliers, and ensures that all models delivered will match together when doing the FE assembly.

Define the modelling requirements the engine provider has to comply to

As shown in Figure 207 below, at this step of the scenario, the PPS mechanical integrator now has to check the table describing the whole mesh assembly structure with the attached files.

Figure 207: PPS mechanical IFEM structure with attached documents and related attributes

In order to send this package to the model providers, the PPS Integrator is going to use a dedicated standard process for simulation data exchanges. He instantiates the template and references the data needed to be sent to the supplier:

- Instantiated simulation template defining the simulation context, inputs and all expected outputs.
- xml describing the mechanical system architecture specified by the integrator (blocks, connectors and ports + associative links between CAD models and blocks and between CAD mating features and ports);
- xml describing the relation CAD-FEM (CAD mating features with FE mesh templates)
- xml file describing the FE modeling requirements and the attached files (publication text file, CAE requirements, Quality report)

The process is run and the whole technical data package is exported and sent to the BDA hub.

13.2.2.3 STEP 2-3: PLM/SLM Interoperability via BDA hub and semantic data mapping

This capability supports the data exchange steps (between steps 2 and 3, between steps 4 and 5 and between steps 7 and 8). The objective is to demonstrate the potential use of a common BDA Business Object Model (BDA BOM) used for heterogeneous PLM/SLM data exchanges. The BDA BOM is the common language developed within Crescendo. In this step, BDA BOM modules have been implemented in a semantic data mapping tool - sDM© (semantic Data Management) developed by Vinci Consulting[8] and illustrated in the middle of Figure 208. The sDM© application is based on a multi-layered (conceptual and logical) and scalable (easy follow-up of applications changes) architecture enabling to define a neutral conceptual meta-model and to map, when it is possible, various business applications to this meta-model.

[8] Official website of Vinci Consulting: http://www.vinci-consulting.com/

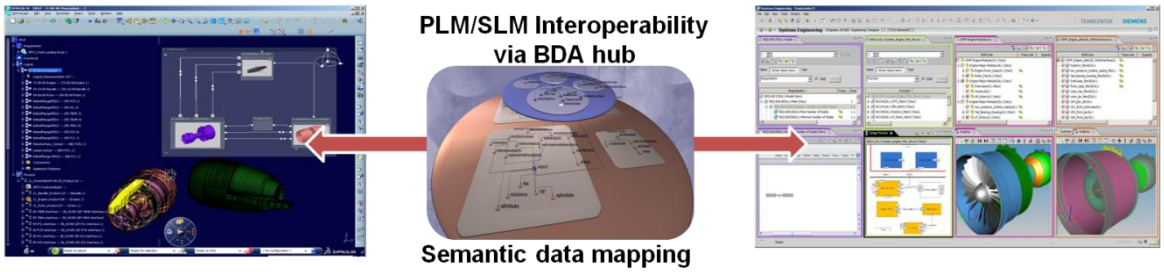


Figure 208: overview of PPI developed capabilities in the commercial PLM solutions

The general methodology to perform this mapping is:

- Prepare the specification package respecting standards and exchange rules defined within the partnership;
- Select a subset of concepts present in the BDA Business Object Model answering the use case needs;
- With the selected concepts, model the conceptual layer. If necessary (if a concept or object does not exist in the BDA BOM), customize the conceptual layer adding the necessary objects/classes;
- Model the applications semantics i.e. the meaning of the information present in respective commercial PLM/SLM tools;
- Describe how an application data model is mapped to the conceptual layer (semantic links between the conceptual layer and the application layer);
- Access to data sources without any modification of their native data model;

In the context of the Product Integration scenario, the BDA Business Object Model (BDA BOM) is the common language or the shared data semantics and sDM© is the connector or mediator enabling PLM applications to understand this language.

1) Model concepts extracted from BDA BOM

The main capability proposed by sDM© is to manage shared data semantics by modelling a neutral and conceptual meta-model on the top of the business proprietary applications. Conceptual and logical data models are all defined in RDF triplets in sDM© so that to exploit the semantic web technologies. The RDF[9] (Resource Description Framework) is a major component in the W3C's Semantic Web activity and is supposed to enable automated software to store, exchange, and use machine-readable information distributed throughout the Web. The RDF triplet is a mechanism for describing resources (mainly web resources) and linking these resources to properties and property values. This mechanism enables to check the quality of the data and aggregate them. In Figure 209, a part of the BDA BOM has been modelled in sDM© which automatically generate a BDA BOM compliant RDF file (transparent for the user) that will serve for mappings other specific logical data models.

[9] http://www.w3.org/RDF/

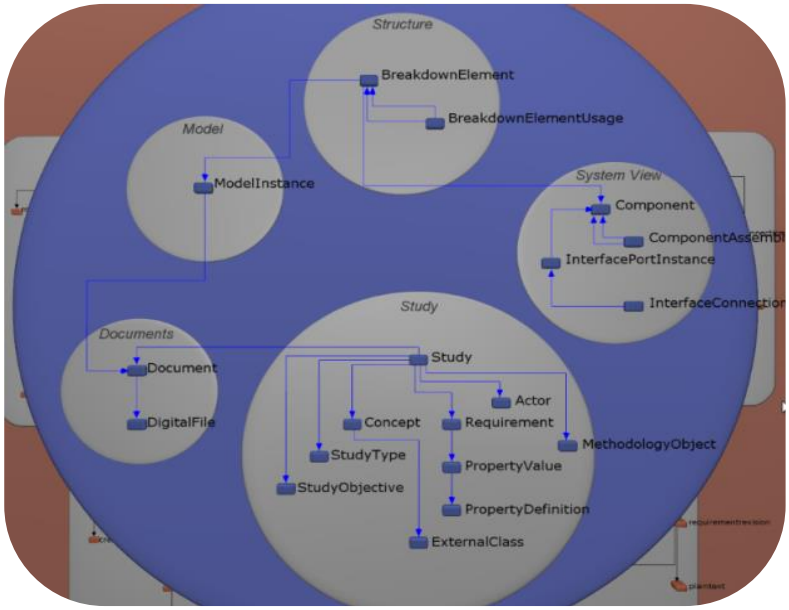


Figure 209: extract of the BDA BOM used for the product integration scenario and modeled in sDM©

2) Model business applications

As shown on Figure 210, sDM© enables to design any kind of data model coming from any kind of business application. In this figure, extract of the Simulia/Catia V6 data models have been modelled as well as the ENVOVIA V6 and Teamcenter for Simulation 9 data models.

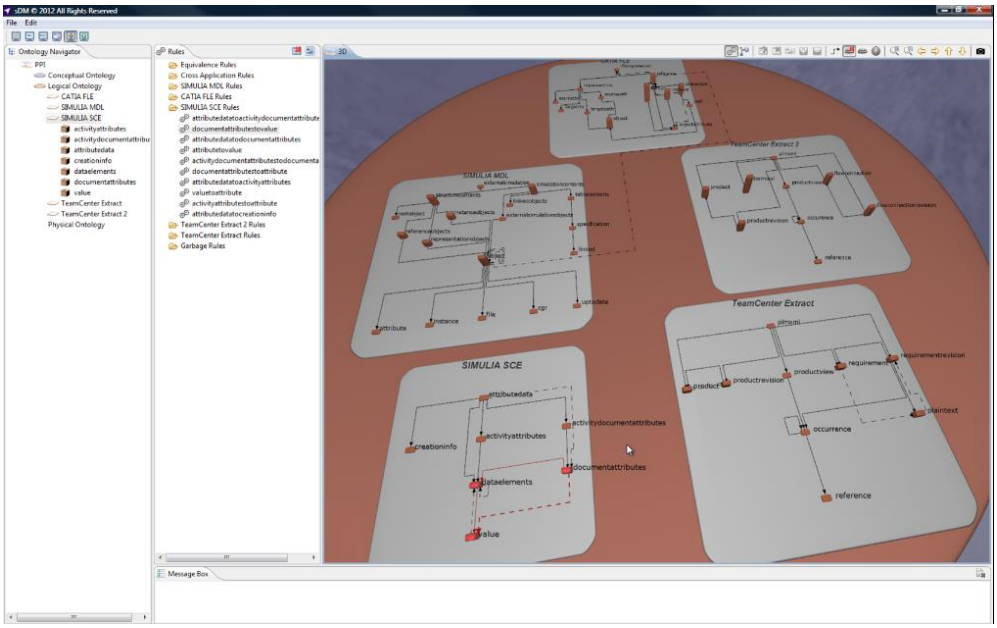


Figure 210: Simulia/Enovia V6 and Teamcenter for Simulation 9 data models modeled in sDM©

3) Map applications to BDA BOM

As shown in Figure 211, the sDM© graphical user interface allows an easy mapping of classes and attributes between the BDA BOM meta-model and the business applications that need to interoperate.

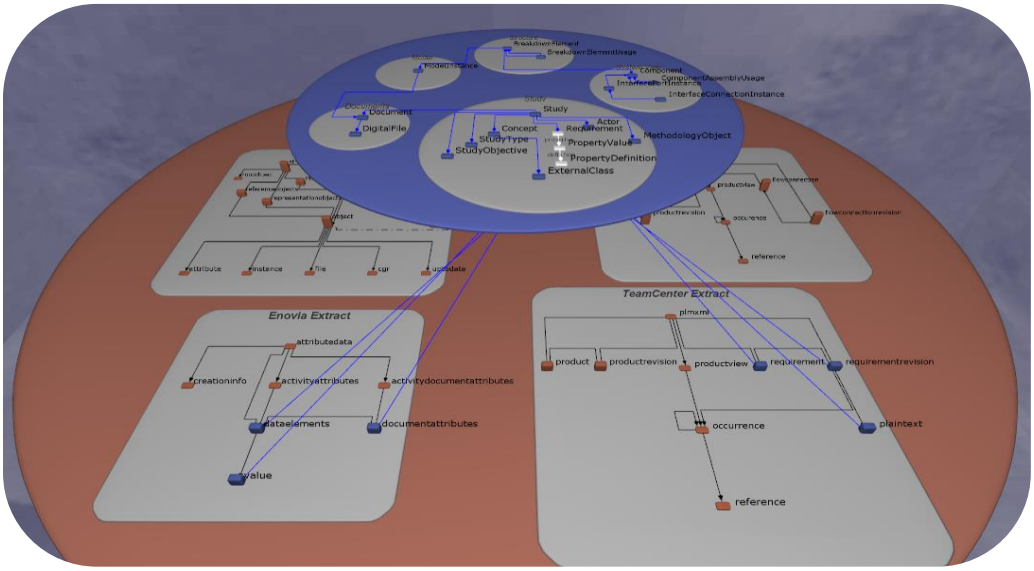


Figure 211: mapping of Enovia V6 and Teamcenter for simulation 9 data models with the BDA BOM in sDM©

Once the mapping defined, sDM© offers the capability to automatically generate the RDF file compliant with the appropriate application data model.

4) Required transformation services

The data exchange sequence and the successive transformations required for the semantic data mapping are detailed in Figure 212 below.

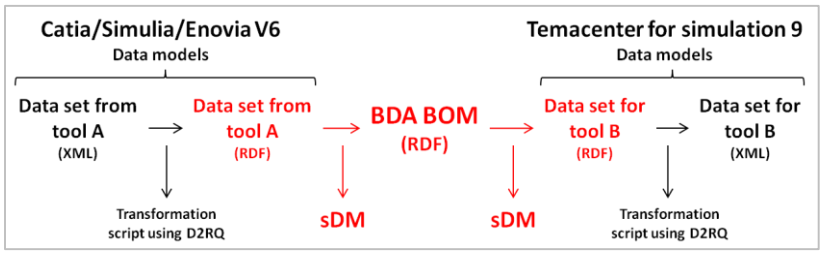


Figure 212: Required transformations for the semantic data mapping with the BDA BOM

The red parts of Figure 212 indicate the scope of sDM© and the scope of the demonstrator presented at the end of the Crescendo project. The BDA BOM meta-model is described in RDF1 triplets. Therefore, input data (XML files generated in STEP 2) need to be translated as RDF1 triplets. This transformation has been performed using the D2RQ Mapping Language[10] which is a declarative mapping language for describing the relation between ontologies expressed in RDF triplets or OWL and relational data models. The transformation of the XML file exported from Enovia V6 into RDF1 triplets is shown in appendix XV. Table 23 and Table 24 shows respectively the XML file representing the PPS logical and behavioural architecture exported from Enovia V6 and its equivalent representation in RDF1 triplets.

13.2.2.4 STEP 3 – Create engine FEM

Once the 3D-XML files exported from ENOVIA V6 have been translated and mapped to the BDA BOM and translated again into a PLM-XML files compliant with Teamcenter for Simulation data model, the engine model provider receives a notification and imports the technical data package containing

[10] Official website of D2RQ platform : http://d2rq.org/d2rq-language/

the study model request with the engine mesh and interfaces modelling requirements (see Figure 213 below).

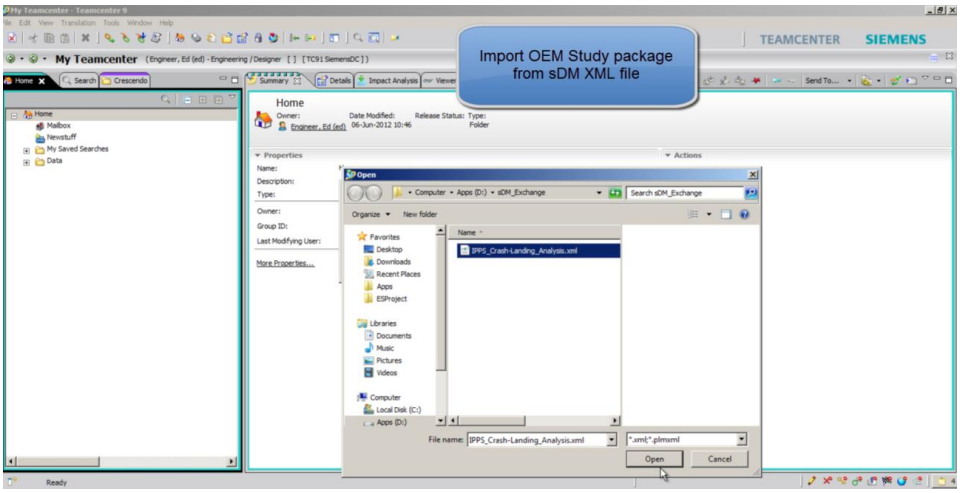


Figure 213: PLM-XML file generated from sDM and imported in Teamcenter for Simulation 9

Table 10 displays the content of the PLM-XML file. Similarly to SIMULIA/ENOVIA V6, where product and simulation data are structured according to the RFLP framework, in Teamcenter for Simulation 9 (TC4SIM9) product and simulation data are structured and displayed according to four viewpoints:

- DMU view (Product view): managing product configurations and 3D CAD data;
- Logical view (System view): managing logical architectures where simulation models and interfaces are specified;
- Functional view (Requirements): managing the whole product and simulation requirements tree and their links with DMU, logical or behavioural items;
- Behavioural view (FEM view): managing BMU FE data (FE models and FEA results) and their links with DMU, logical and requirements items.

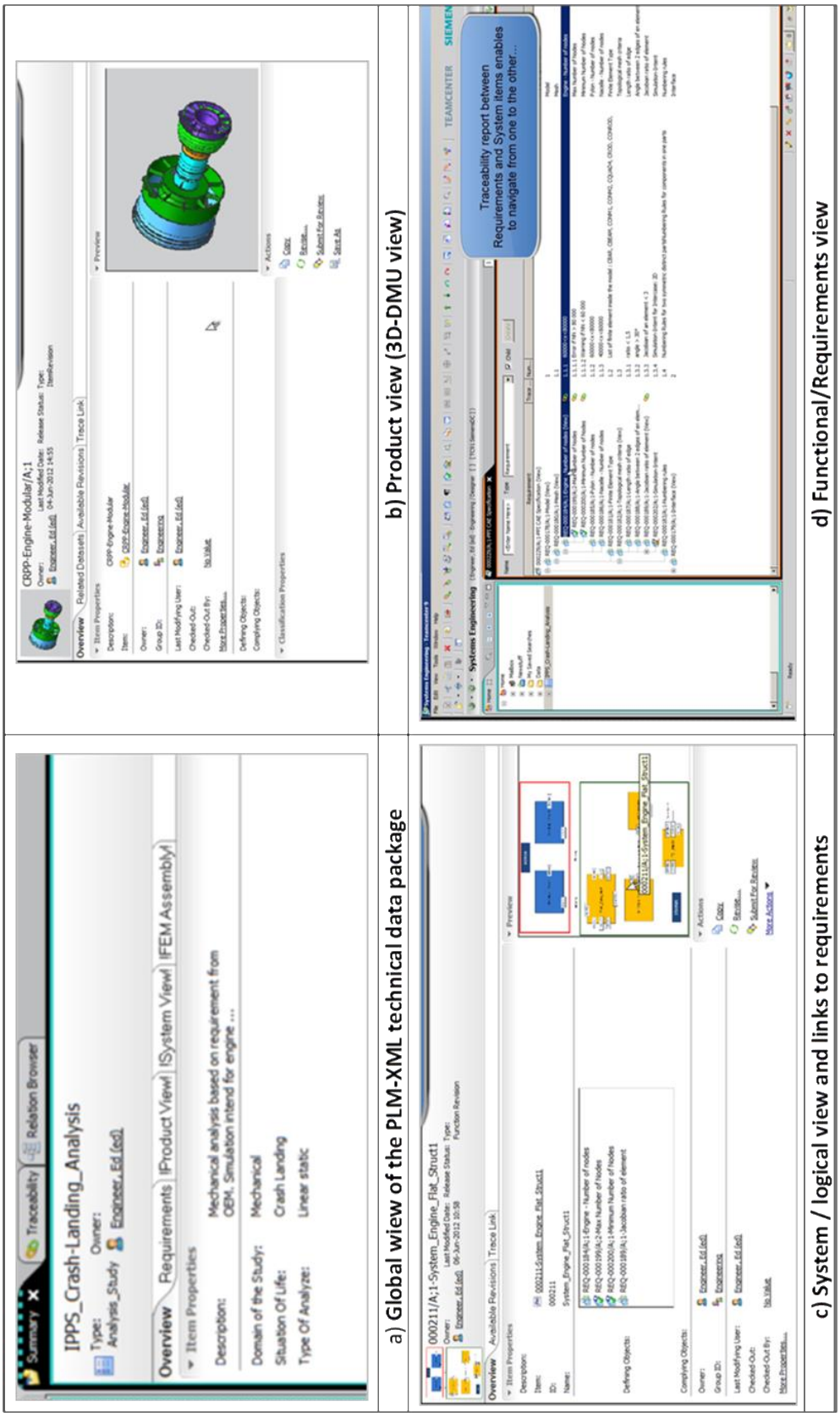


Table 10: Content of the PLM-XML technical data package - Global view (a), Physical view (b), System/Logical view (c) and Fucntinal/Requirements view (d)

From the requirements perspective, the engine integrator opens the logical mechanical architecture of the engine and its sub-systems and components as shown in Figure 214 below.

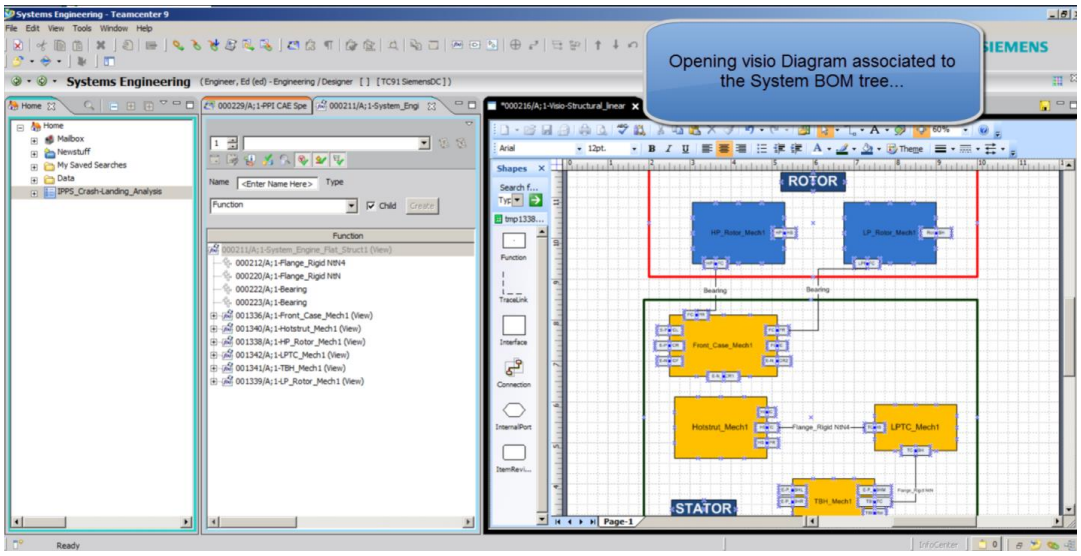


Figure 214: Engine mechanical logical architecture managed with an integrated Visio module in TC4SIM 9

The integrator realises that the intercase mesh model is missing to provide the appropriate engine IFEM. Moreover the logical architecture is not complete and some interfaces are not yet specified. First the engine integrator needs to complete its mechanical architecture specifying the CAD mating features between engine sub-systems and link these mating features to the appropriate ports in the system diagram. Like in step 2 for the PPS integrator, the engine model provider acts as the engine model integrator and has to specify CAD mating features and mechanical connections and their association with their corresponding logical items. Figure 215 shows how the CAD mating features are defined between the hot strut and the intercase models.

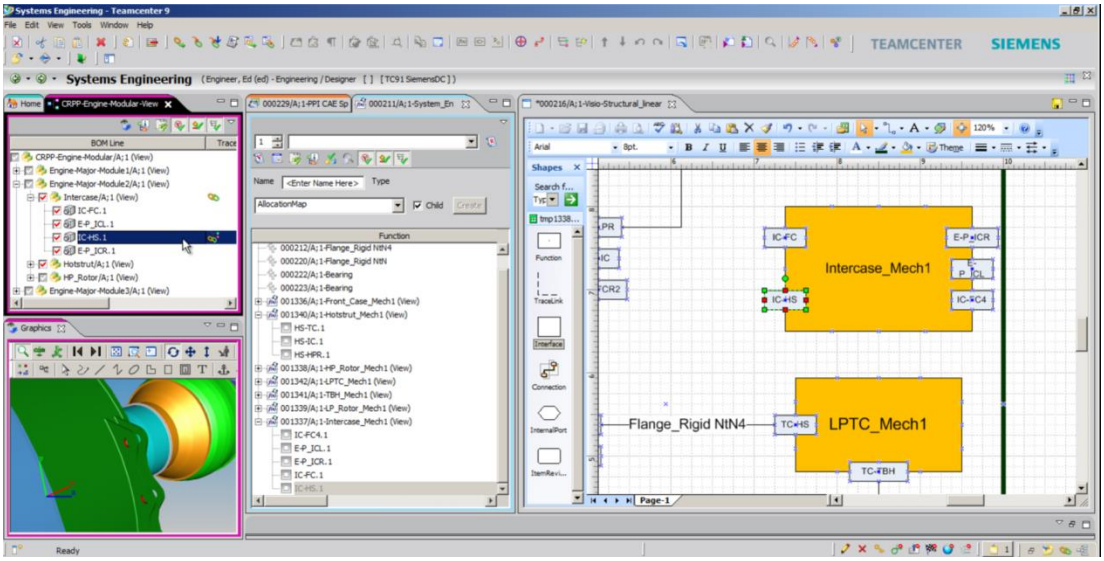


Figure 215: Referencing and Associating Ports-CAD Publications links in TC4SIM 9

Similarly to the solution prototype proposed in CATIA/SIMULIA V6 and as shown in Figure 216, TC4SIM also proposes an interface catalog/library which contains standard connections for specific simulations. The methodology remains the same:

- Specify the connection attributes (domain, type, technology) to search in the catalog;
- Access and retrieve the appropriate FE connection template in "interface catalog"
- Instantiate each FEM template and synchronize with system parameters
- Populate the parameters of each system connection

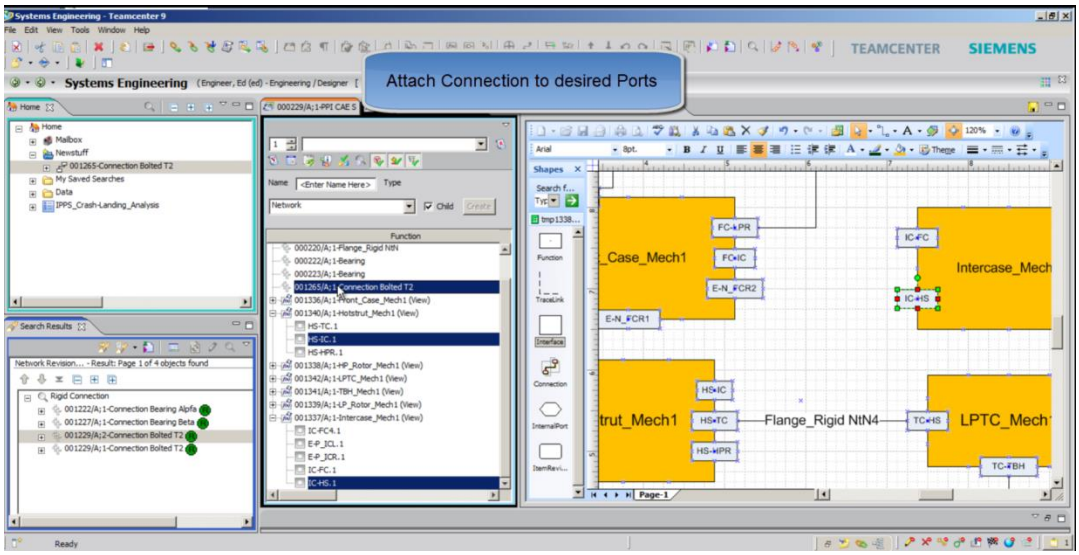


Figure 216: Setting-up mechanical connections from catalog to the system view in TC4SIM

After having completed the mechanical architecture of the requested engine FEM, the engine integrator has to cascade the CAE modelling requirements (simulation intent) and send a simulation model request to the intercase model provider. First the engine model integrator needs to attach a specific set of modelling requirements (simulation intent) to the added Intercase component occurrence used in the IFEM architecture previously completed (see Figure 217 below).

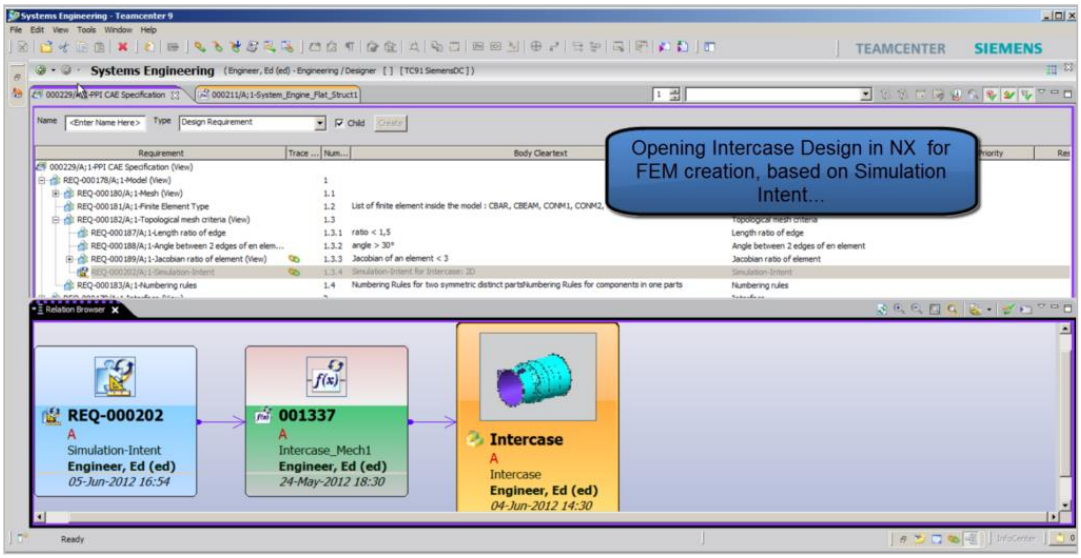


Figure 217: TC4SIM relation browser for visualising dependency links between simulation intents, meshes and CAD models

In this step, and thanks to the collaboration and support of the Aerospace and Manufacturing Research Cluster of Queen University of Belfast and in particularly thanks to the work of [Nolan et al., 2013], we demonstrate the use of simulation intents and templates (including simulation context and associated models and interfaces modelling requirements) for performing automatic fit-for-purpose meshing. The whole process performed and automated in NX-CAE is illustrated in Table 11.

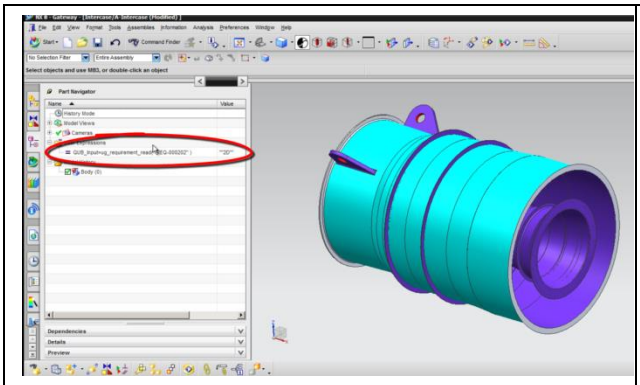
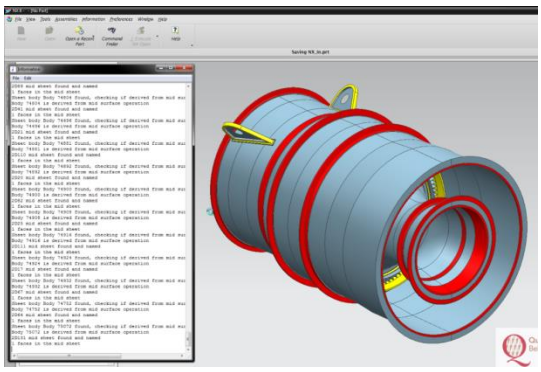
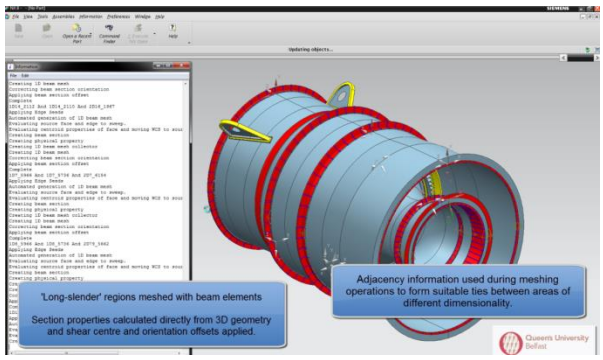
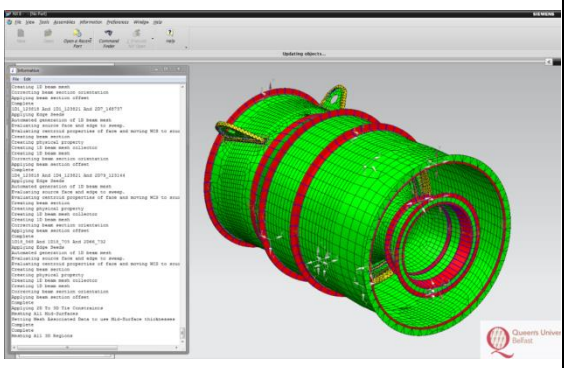
	
Idealised CAD model used for the mesh creation	**"Thin" regions transformed in mid-surface elements**
	
Long-slender regions transformed into beam elements	**Remaining thin regions transformed into shell elements and complex regions into tetrahedral elements**

Table 11: automatic "fit-for-purpose" meshing process of the intercase thanks to a script developed by [Nolan et al., 2013]

The associative model network – displayed here in the TC4SIM relation browser – is then enriched with new generated models (see Figure 218 below).

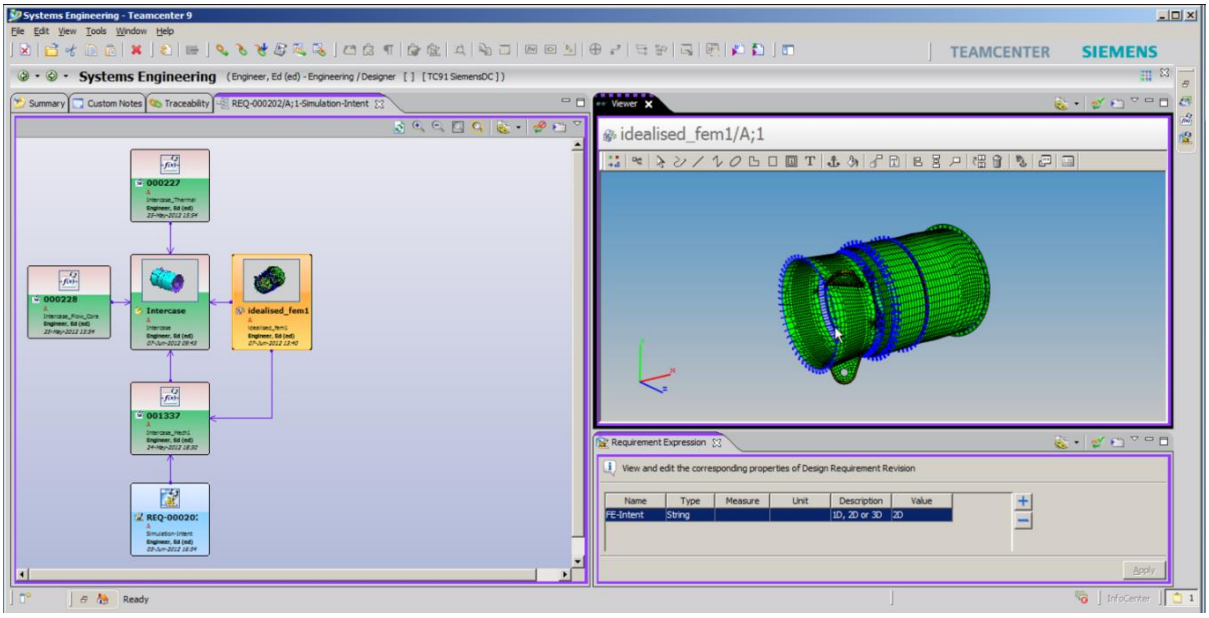


Figure 218: Associative model network implementation – the relation browser or traceability report in TC4SIM 9

Dependency links are created between the mesh model and the simulation intent (modelling requirements), between the mesh model and the idealised CAD model. We can notice that another simulation intent (or thermal analysis) is also linked to the idealised CAD model but not to the mesh.

Finally, after having retrieved all the required FE models, the engine mechanical integrator call a specific NX-CAE script to automatically create FE links objects and to integrate the engine components FE models. The result is shown in Figure 219 below.

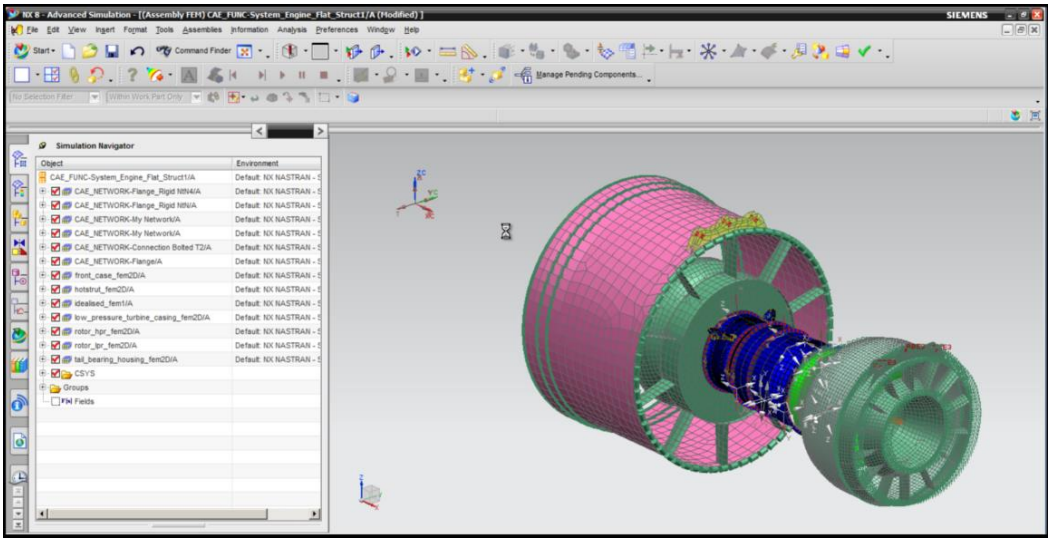


Figure 219: script automating the IFEM assembly - creation of NX-CAE FE links objects

13.2.2.5 STEP 4 – Assess and check models quality/reliability

Generated and provided simulation FE models are gathered from several sources and the integrator has no control on the reliability of the data for the simulation to perform. Due to this statement, some models imply approximate results that can drive to false interpretations about the product behaviour prediction. In order to ensure the reliability of the model, two main tests are required:

- Ensure the reliability of models by modelling requirement verification;
- Accuracy assessment of simulation model.

This step demonstrates how an automated quality check procedure can be set-up to generate models quality criteria and hence to assess reduced models reliability and to provide information concerning the potential accuracy of integrated analysis' results. We propose to set-up a virtual testing of the component model with a light simulation process (e.g. static linear analysis) as shown in Figure 220 below.

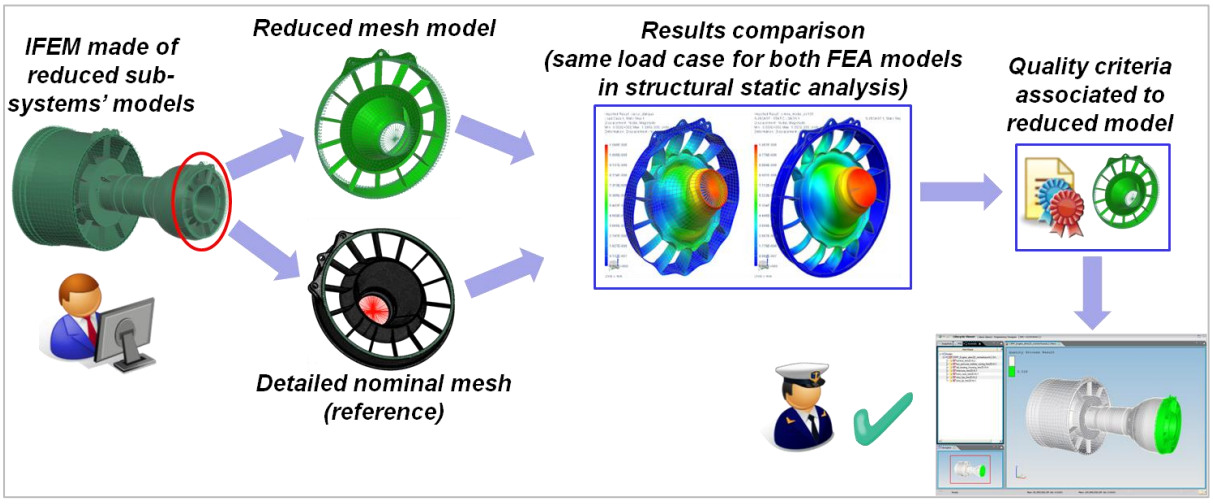


Figure 220: illustration of the FE model quality check procedure

This light simulation process is applied on two different meshes:

- A reference mesh with 3D finite element, directly based from the original CAD files;
- The mesh to check is created manually and based on behaviour assumptions (use of shell, beam or mass elements).

The goal is to compare the result of the analysis of the two meshes and give a confidence rate about the good representation of the behaviour of the product regarding the modelling assumptions. The comparison has to give information on two levels of the results. The first level stands for evaluation of global results on the entire mesh. The second stands for local evaluation of the meshes. A correlation has been done between the two meshes, the reference mesh and the mesh to check. The comparison of the results provided by the two meshes has to be transparent for the user and be driven by an automatic workflow as illustrated in Figure 221.

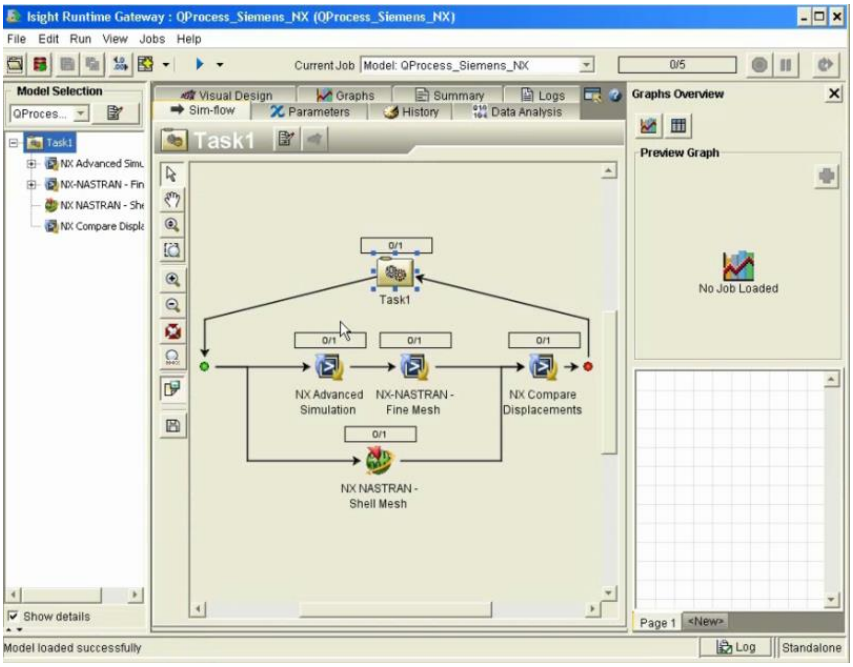


Figure 221: Automated FE model quality check workflow modelled and configured in Isight Design Gateway

The user has only to initiate the simulation and to define the local area inside the meshes to make the local comparison. Based on these comparisons, a rate of confidence will be applied. This rate gives a level of confidence in the accuracy of the results provided by the mesh (see Figure 222 below).

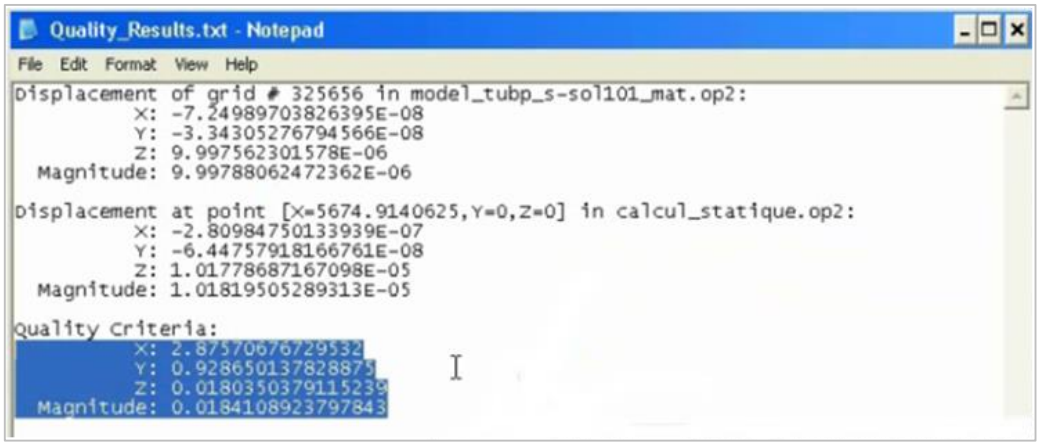


Figure 222: output of the workflow executed in batch - text file containing displacement results and their comparison

TC4SIM 9 also proposes 3D visual report capabilities to visualise the quality ratio on the whole integrated FE model (see Figure 223 below).

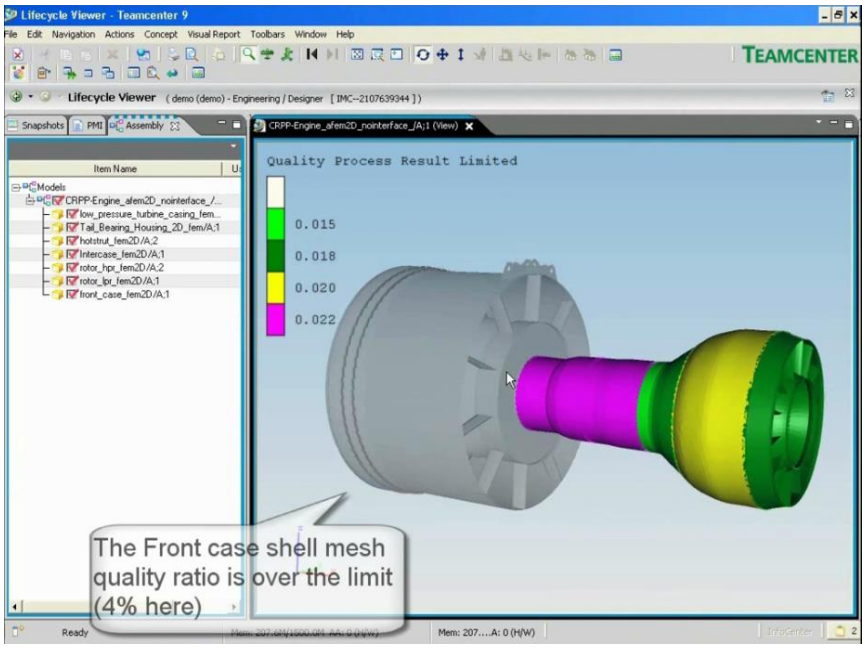


Figure 223: visualisation of quality ratio (results accuracy) on the integrated engine FEM

The user can also use these 3D visual reports to visualise and check the compliance of the mesh features with its simulation intent and related FE modelling requirements as shown in Figure 224 where the engine model provider checks the values of the jacobian ratio of all engine constituting FE models.

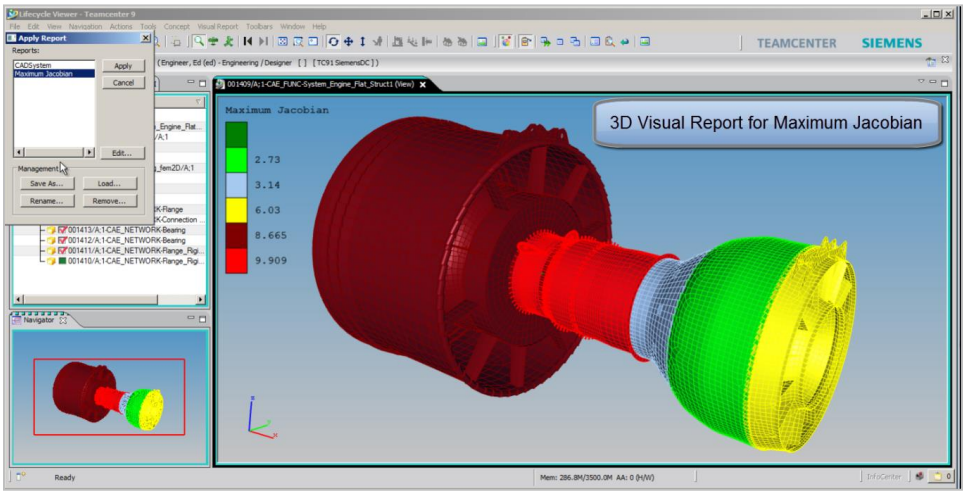


Figure 224: 3D visual report after the maximum jacobian ratio checking procedure

The comparison result and the confidence ratio need to be managed in the SLM data models in order to ensure the traceability by associations between the mesh, the simulation intent and the results of the quality check procedure.

The assessment of the whole engine FEM to provide is now finished. The technical data package, containing the engine IFEM and its quality report, is ready to be sent back to the PPS mechanical integrator. The data exchange sequence is performed similarly to the first one (see 13.2.2.3) but in the opposite direction.

Figure 225 synthesises how the "integrator dedicated environment has been implemented in TC4SIM 9. Whereas, in CATIA/SIMULIA V6 the data are managed and organised following the RFLP data structure, here TC4SIM 9 provides a similar multi-view integrator environment where functional, logical, structural and behavioural product data and viewpoints of a same product or system can be managed in parallel. This enables to easily define dependency links between functional, structural and behavioural features of the studied system (if the data model permits it).

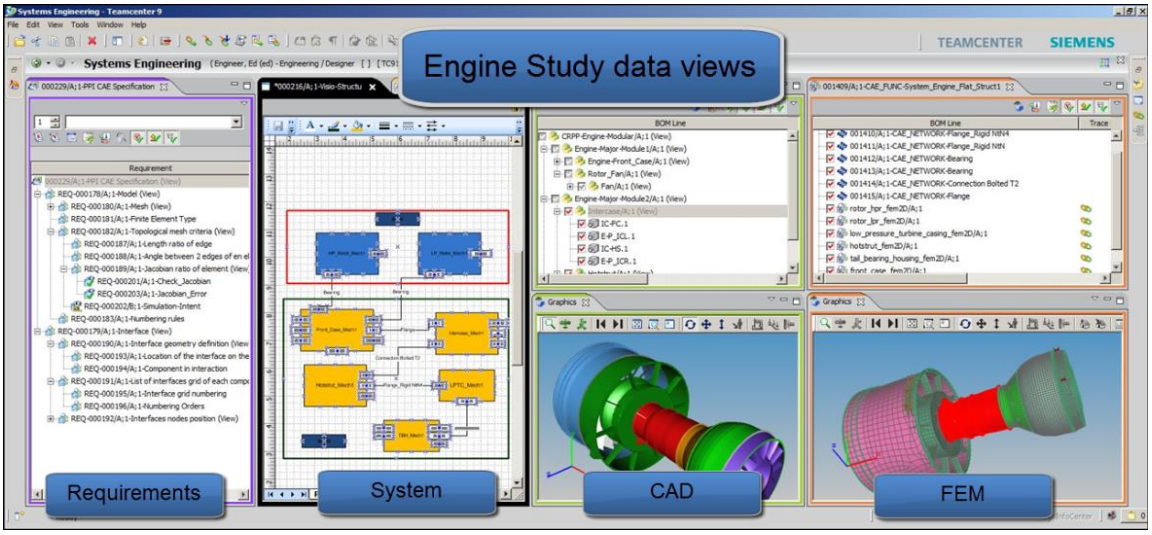


Figure 225: Completed multi-view integrator environment in TC4SIM 9

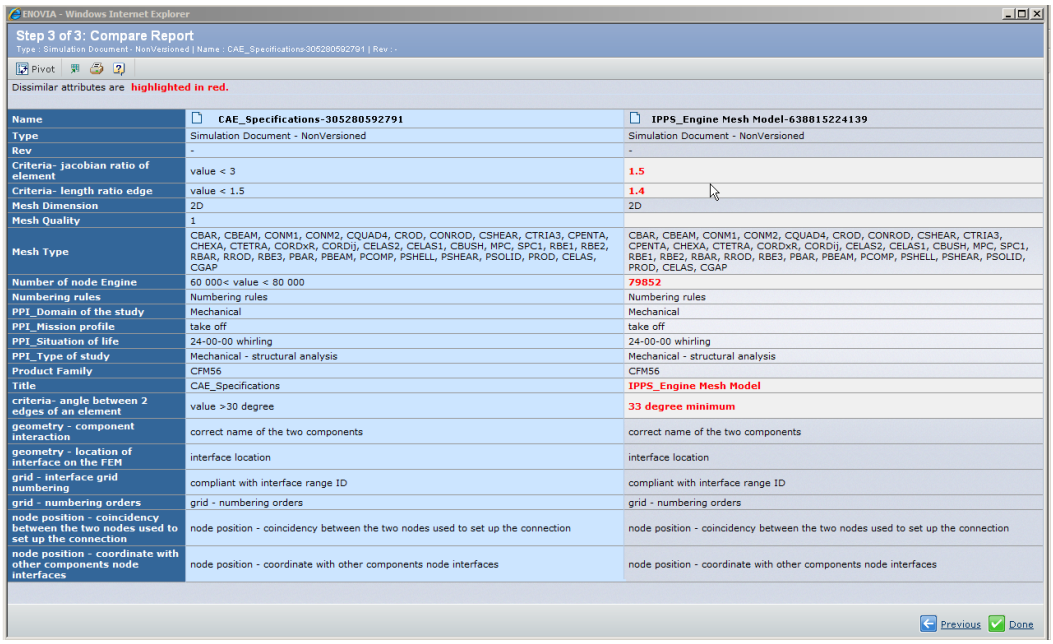
13.2.2.6 STEP 5: Check and integrate sub-systems FE models

Once the PPS sub-systems FE models provided by the sub-systems integrators and before integrating these models, the study chief engineer and final integrator has to check the compliance of these FE models features with the modelling requirements he specified in STEP 2. The method for checking the compliance of FE models features with specified modelling requirements consists of setting-up some specific attributes to describe the mesh. These attribute will be like the identity card of the mesh:

- The description of the simulation for which the mesh is used;
- The modelling requirements of the mesh depending of the type of the simulation;
- The mesh characteristic and metric parameters;
- Nodes numbering rules;
- Number of elements and nodes;
- Interface modelling requirements for assembly;
- Topological requirements based on boundaries for mesh measures (Jacobian, aspect ratio, mesh area, etc.)

The mesh characteristics stand for all attributes which measures the mesh topology with its different characteristics (Mesh type, number of nodes, Jacobian ratio, aspect ratio...). All these attributes are associated to the FE model or gathered in a quality report associated to the FE model. In any case, these attributes have to be accessible within the PLM/SLM environment.

Back in the ENOVIA V6 environment, and as shown in Figure 226, the PPS integrator calls an automated task providing a comparison table between the CAE modelling requirements and the corresponding FE models features. The non-compliant features are highlighted in red.


ENOVIA - Windows Internet Explorer

Step 3 of 3: Compare Report

Type : Simulation Document - NonVersioned | Name : CAE_Specifications-305280592791 | Rev : -

Pivot

Dissimilar attributes are highlighted in red.

Name	CAE_Specifications-305280592791	IPPS_Engine Mesh Model-638815224139
Type	Simulation Document - NonVersioned	Simulation Document - NonVersioned
Rev	-	-
Criteria- jacobian ratio of element	value < 3	1.5
Criteria- length ratio edge	value < 1.5	1.4
Mesh Dimension	2D	2D
Mesh Quality	1	
Mesh Type	CBAR, CBEAM, CONM1, CONM2, CQUAD4, CROD, CONROD, CSHEAR, CTRIA3, CPENTA, CHEXA, CTETRA, CORDxR, CORDij, CELAS2, CELAS1, CBUSH, MPC, SPC1, RBE1, RBE2, RBAR, RROD, RBE3, PBAR, PBEAM, PCOMP, PSHELL, PSHEAR, PSOLID, PROD, CELAS, CGAP	CBAR, CBEAM, CONM1, CONM2, CQUAD4, CROD, CONROD, CSHEAR, CTRIA3, CPENTA, CHEXA, CTETRA, CORDxR, CORDij, CELAS2, CELAS1, CBUSH, MPC, SPC1, RBE1, RBE2, RBAR, RROD, RBE3, PBAR, PBEAM, PCOMP, PSHELL, PSHEAR, PSOLID, PROD, CELAS, CGAP
Number of node Engine	60 000< value < 80 000	79852
Numbering rules	Numbering rules	Numbering rules
PPI_Domain of the study	Mechanical	Mechanical
PPI_Mission profile	take off	take off
PPI_Situation of life	24-00-00 whirling	24-00-00 whirling
PPI_Type of study	Mechanical - structural analysis	Mechanical - structural analysis
Product Family	CFM56	CFM56
Title	CAE_Specifications	IPPS_Engine Mesh Model
criteria- angle between 2 edges of an element	value >30 degree	33 degree minimum
geometry - component interaction	correct name of the two components	correct name of the two components
geometry - location of interface on the FEM	interface location	interface location
grid - interface grid numbering	compliant with interface range ID	compliant with interface range ID
grid - numbering orders	grid - numbering orders	grid - numbering orders
node position - coincidency between the two nodes used to set up the connection	node position - coincidency between the two nodes used to set up the connection	node position - coincidency between the two nodes used to set up the connection
node position - coordinate with other components node interfaces	node position - coordinate with other components node interfaces	node position - coordinate with other components node interfaces

Previous Done

Figure 226: Modelling requirements compliance checking - comparison of FE features with FE modlling requirements

Once all requested meshes have been provided, checked and considered enough reliable to be integrated by the PPS mechanical integrator, they can now be assembled. The mechanical logical architecture designed for the creation of the PPS mechanical IFEM need to be completed by assigning the provided FE models to the appropriate system representation elements (blocks, ports and connections). The behavioural parameters and representations of the interactions between FE models are hence specified within the logical architecture describing the PPS IFEM mechanical architecture defined in STEP 2. This architecture is used to perform automatically the assembly of these sub-systems FE models. To automate the assembly task, Dassault Systèmes has developed a specific script (see Figure 227) enabling to translate the architecture of the IFEM described in the system view (Logical architecture in the RFLP tree) into the language of the appropriate pre-processing tool (here Abaqus CAE), in order to compute it with the appropriate specified solver.

```
from abaqus import *
from abaqusConstants import *

import os

extensionBDF = ".bdf"
extensionDAT = ".dat"

filenames = os.listdir('.')
content = ''
for f in filenames:
        if (f.endswith(extensionBDF)) or (f.endswith(extensionDAT)):
                content = content + '\n' + open(f).read()
                open('IPPS-CrashLanding01.dat','wb').write(content)

cwd = os.getcwd()

session.Viewport(name='Viewport: 1', origin=(0.0, 0.0), width=396.327056884766,
    height=190.147216796875)
session.viewports['Viewport: 1'].makeCurrent()
session.viewports['Viewport: 1'].maximize()

from caeModules import *
from driverUtils import executeOnCaeStartup
executeOnCaeStartup()
session.viewports['Viewport: 1'].partDisplay.geometryOptions.setValues(
    referenceRepresentation=ON)
session.viewports['Viewport: 1'].assemblyDisplay.setValues(
    optimizationTasks=OFF, geometricRestrictions=OFF, stopConditions=OFF)

mdb.ModelFromNastranInputFile(
    inputFileName=cwd+os.sep+'IPPS-CrashLanding01.dat',
    modelName='IPPS-CrashLanding01', cbar='B31', cquad4='S4R', chexa='C3D8I',
    ctetra='C3D10M', sectionConsolidation=PRESERVE_SECTION,
    preIntegratedShell=False, weightMassScaling=True, loadCases=True,
    coupleBeamOffsets=True, keepTranslatedFiles=True)

a = mdb.models['IPPS-CrashLanding01'].rootAssembly
session.viewports['Viewport: 1'].setValues(displayedObject=a)
```

Figure 227: script automating the assembly of the PPS mechanical IFEM in Abaqus CAE

An activity for integration is created from template. This activity uses the script of Figure 227 to merge all meshes. As shown in Figure 228, the script launches Abaqus CAE in batch mode, the FE models (INP files) are automatically imported, the FE links object (defining the connections between the FE models) automatically created and the nodes with same numbering and location merged.

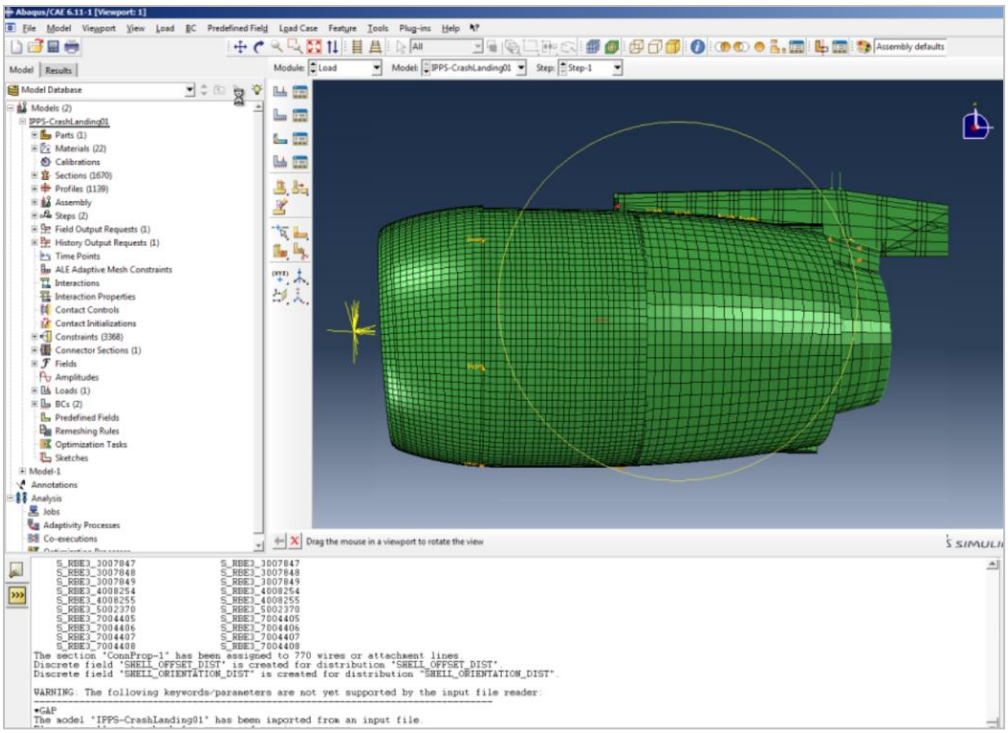


Figure 228: script automating the IFEM assembly - creation of Abaqus FE links objects

Now the global power plant assembly is created and this file can be saved in the PLM/SLM data base.

13.2.2.7 STEP 6 – Prepare the IFEM and perform simulation

As shown on Figure 229, the generated PPS IFEM is now integrated in the RFLP tree and associated to its corresponding logical architecture and to the DDMU from which it was derived.

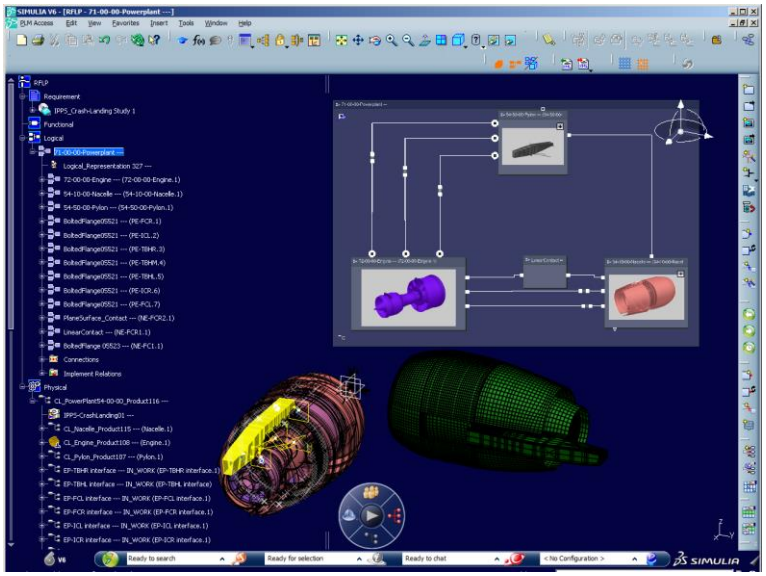


Figure 229: DDMU and corresponding BMU integrated in the RFLP environment

The assembly model is ready to be computed: the boundary condition are applied on interfaces and the engineer can launch the simulation through an activity template gathering all required data (IFEM, boundary conditions and load cases) and launching the solver in batch mode again. Once it is executed, the global results (result files and reports) can be visualized (see Figure 230 below) and post-processed (analyzed).

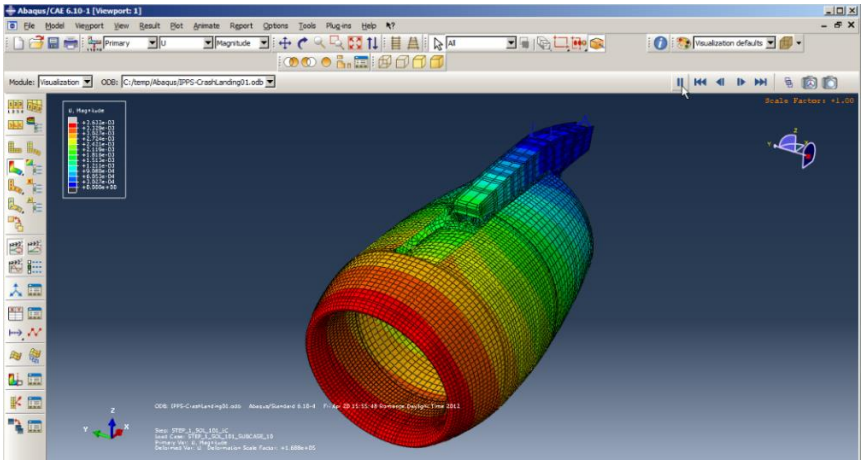


Figure 230: simulation execution and results visualisation in Abaqus CAE

The crash landing simulation is done. The post process can be started. If the results are satisfying and exploitable, they will be distributed to models suppliers in order to perform finer simulations at sub-systems and parts levels.

13.2.2.8 STEP 7 – Exploit and distribute simulation results

In this step we demonstrate how a standard dedicated and automated process can be used to:

- dispatch the displacements, loads and strains at the components interfaces in a file;
- attach the file to the appropriate ports in system view;
- Use the system/logical view as a decision support environment to display status of results compared to requirements.

The results are retrieved from an execution activity and added to extraction activity. Similarly to previous execution the user splits the global result file into several files for each interface. These files represent what happens at the interfaces, between the modules. Now, they can be attached to the appropriate ports in the logical/system view. In order to provide a better idea of what is happening at assembly level, an activity maps these results on attributes of the ports so that it is possible to navigate on them by clicking on the ports (see Figure 231).

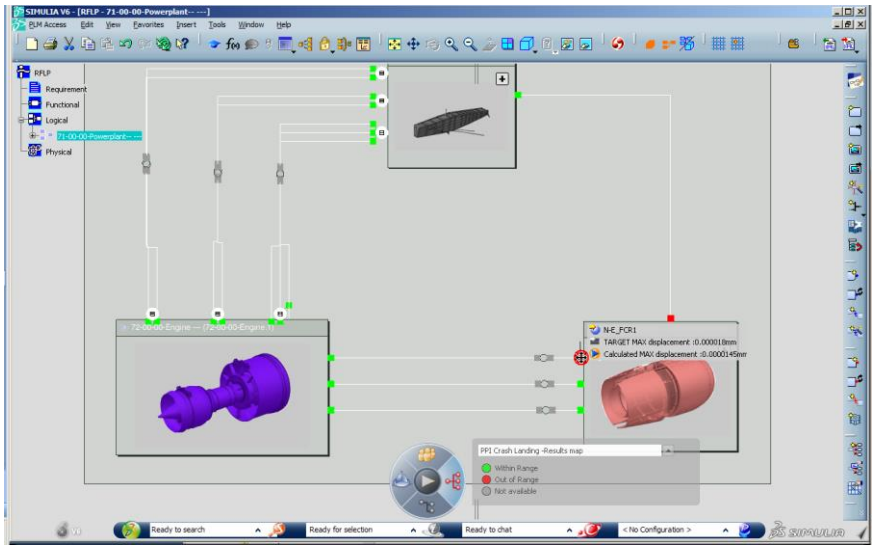


Figure 231: the logical view of the integrator dedicated environment used as a decision support tool for interface results validation in CATIA/SIMULIA V6

The navigation on results displays also the targeted range values derived from design requirements (Requirements/Functional artefacts). This will allow non specialists to better understand the behaviour of the systems for each scenario, and observe directly where there is an issue.

As shown in Figure 232, the integrator now uses another standard process to distribute the displacements at the corresponding interface to suppliers. It is similar to the export activity executed at the end of STEP 2 for packaging the modelling requirements to send to the different models providers. The displacement results at the interfaces can be extracted in a XLS or text file and the values are extracted and defined as interfaces/ports attributes.

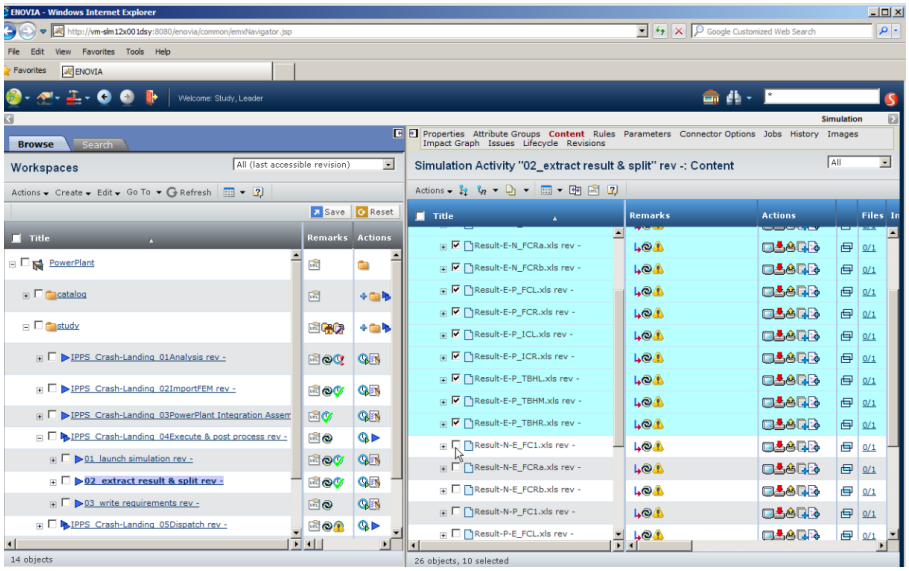


Figure 232: interfaces results extraction in ENOVIA V6

The xml files describing the logical architectures of the PPS sub-systems are now enriched with new interface loads to apply. These interface loads are now packaged with the appropriate xml file and exported to be mapped with the BDA BOM and transformed into PLM-XML files understandable by TC4SIM 9.

13.2.2.9 STEP 8 – Exploit cascaded results

In this step it is shown how the engine integrator (engine model provider) retrieves the interface loads derived from the previous IPPS crash landing simulation and automatically applies them to its engine FEM in order to perform a more accurate analysis at the engine level. This sequence is driven by a process template and its corresponding workflow. Within this workflow an activity is waiting for new loads to apply on engine external interfaces. Once the new interfaces loads imported, a routine enables to map these loads to the appropriate interface ports (thanks to ports naming references in the PLM-XML file) on the engine mechanical architecture that has served the engine FEM generation. Finally, the workflow launches NX-CAE in batch mode and a routine permits to automatically apply the new loads on engine FEM interfaces. The whole process is illustrated in Figure 233. This is the last step of the Product Integration scenario demonstrator.

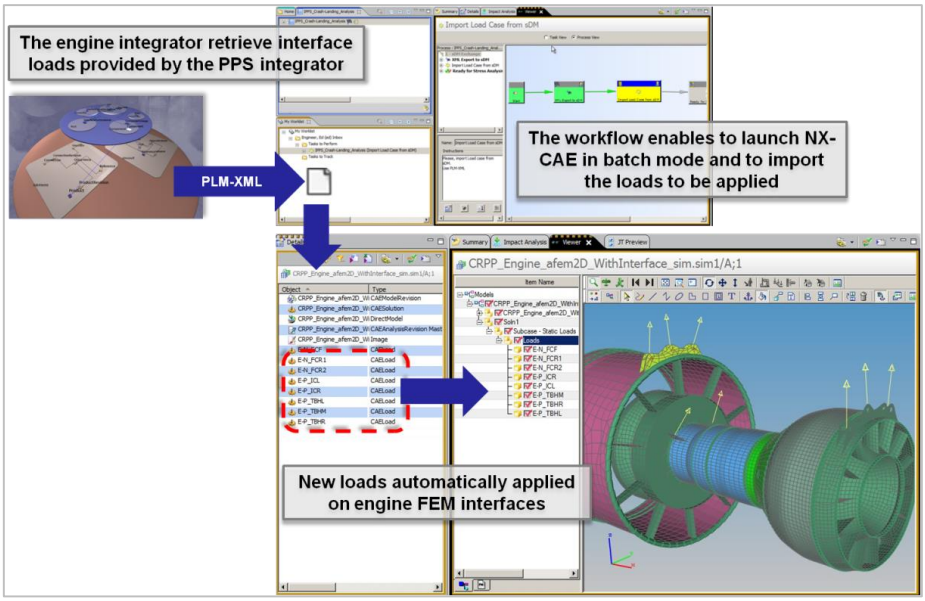


Figure 233: Engine interfaces loads automatically imported in TC4SIM and applied on engine FEM interfaces in NX-CAE

13.3 Contribution to the BDA BOM

The whole BDA architecture framework is described and explained in appendix XVI as well as the BDA information model hierarchy and mapping principles.

If the BDA is to be implemented as an open and modular architecture, information models should be specified and based on standards. In particular, the following has been defined, with reference to the related layers of the BDA architecture framework:

- **The Business Concept model** i.e. the information requirements identified during the development of the processes that are to be supported by the BDA;
- **The Business Object model** i.e. the information that is created or consumed by BDA functions and services;
- **The BDA data model** i.e. the information that is managed by the BDA components.

The Business Object Model (see appendix XVI and Figure 256) is positioned in the Functional Layer of the BDA architecture framework. The Business Object Model is derived from the Business Concept model (and influenced by the process models) and is also represented as an information model, but might have another structure. The BDA data model is mainly based on STEP-AP239 (Product Lifecycle Support – PLCS) and STEP-AP233 (Systems Engineering) standards.

In the context of the product integration scenario demonstrator, and based on the test case introduced in section 10.3.4.2 and Figure 150: the LPTC-TBH assembly test case), this section aims at translating this proof of concept in the BDA BOM semantic. The purposes of this activity are:

- To better understand how the BDA BOM can be used in the frame of the product Integration scenario;
- To see if the BDA BOM enables to manage interfaces like we would like to do in the frame of the Product Integration scenario;
- To have a meta-model describing a common way of managing behavioural (e.g. mechanical) interfaces and a common way of structuring the related data;
- To identify the missing objects/links and complete the data model so that it can answer our expectations;
- To implement this meta-model in sDM (in a more generic way) to be able to perform the mapping sequence between heterogeneous PLM/SLM systems.

13.3.1 System, CAD and FEM interfaces specifications in the BDA BOM

Figure 234 introduces an example of a simple assembly (LPT Case – Tail Bearing Housing) and its representation in the system framework. This figure describes which data need to be associated to the different system framework's objects. From this basic test case we built the corresponding meaning in the BDA BOM semantic (see Figure 234). The assembly and interface constituents are presented in Figure 234:

- The assembly and its constituents (*Component* and *ComponentAssemblyUsage*);
- The interface objects (*InterfacePortInstance*, *InterfaceConnectionInstance*);
- The corresponding CAD models (*GeometryModelInstance*);
- The published faces which are in interaction (*AccessibleModelInstanceConstituent*) attached to the corresponding ports);
- The meshed surface in interaction imposed by the integrator (1st solution for mesh specification) which is a model (the green *GeometryModelInstance* attached to the *InterfaceConnectionInstance*).

This example is for a mechanical interface between the Low Pressure Turbine Case, and the Tail Bearing Housing. It shows a *ConnectionInstance* with property values that define how the connection is to be modelled. The *ConnectionType* has the definitions for the properties, and the image shows the different options for the connections as the different choices are made for the properties. The two ports point to the same *InterfaceSpecification*, and this contains details of the geometry and node locations for the interface. The node locations are provided either by a text document, or as a Nastran representation.

The Physical View is also shown, with the *Components* having *ModelInstance* representations, and the ports having representations of *AccessibleModelInstanceConstituents* inside the *ModelInstances*. The *Connection* also has a *ModelInstance* representation.

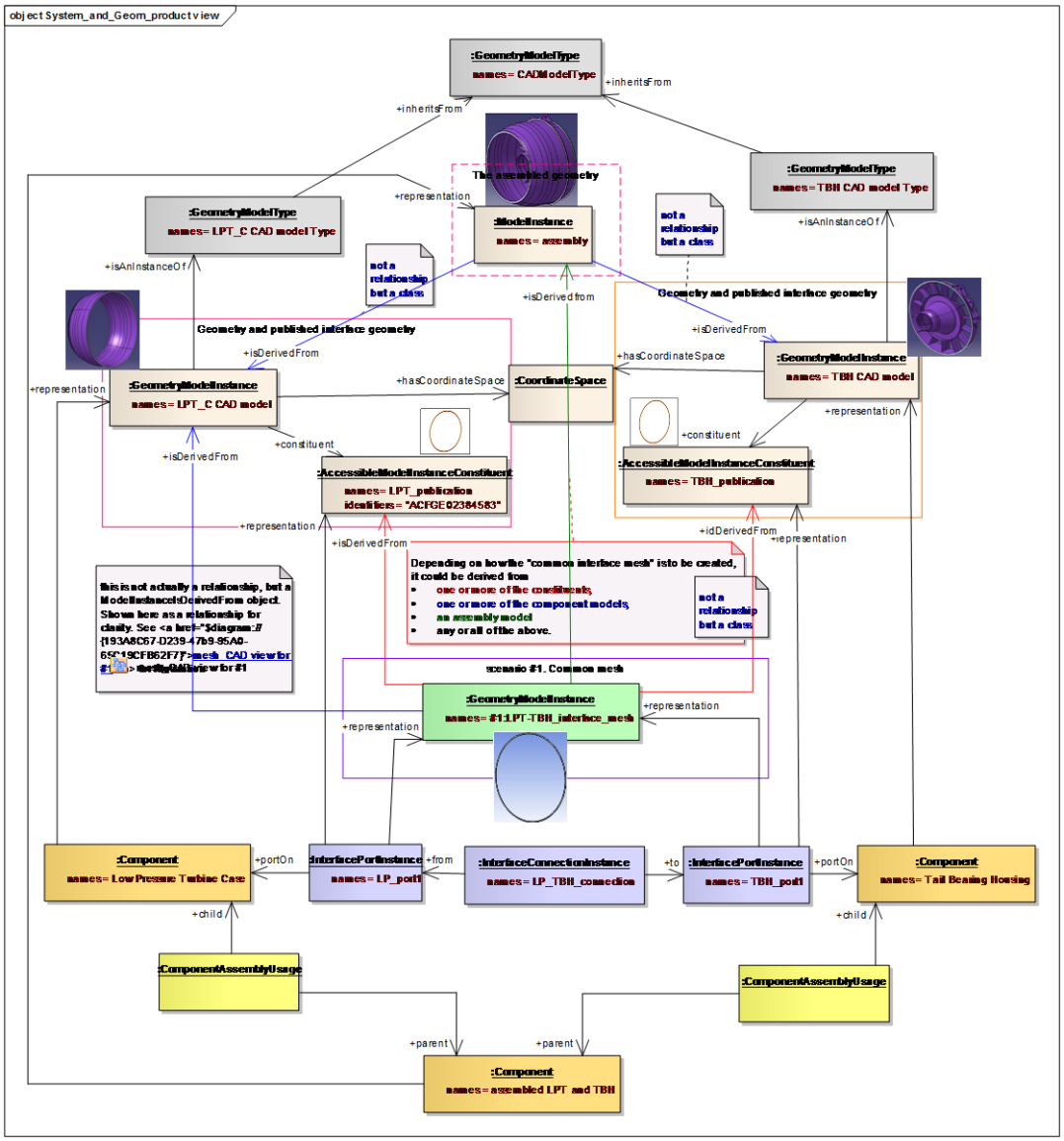


Figure 234: The interface definition via the system framework in the BDA BOM semantic

Figure 235 shows the same system description but considering the 2nd solution for mesh specification: the mesh specification is done specifying the nodes list (number and position/coordinates) that have to match at the interface of the two components' meshes. In that case the mesh specification (*InterfaceSpecification* attached to *InterfaceConnectionInstance*) could be for example a step AP209 file (or other formats supported by the main pre-post tools) created by the integrator and providing the nodes list (*DigitalFile* attached to *Document* attached to *InterfaceSpecification*).

Each of the interface elements in the system view can have zero to many "representations" in the physical view. The class of representation allowed depends on the interface class as follows:

- *InterfaceConnectionType - ModelType*;
- *InterfacePortType - ModelType* and/or *AccessibleModelTypeConstituent*;
- *InterfaceConnectionInstance - ModelInstance*;
- *InterfacePortInstance - ModelInstance* and/or *AccessibleModelInstanceConstituent*;
- *Component - ModelInstance*;
- *ComponentAssemblyUsage - ModelInstance*.

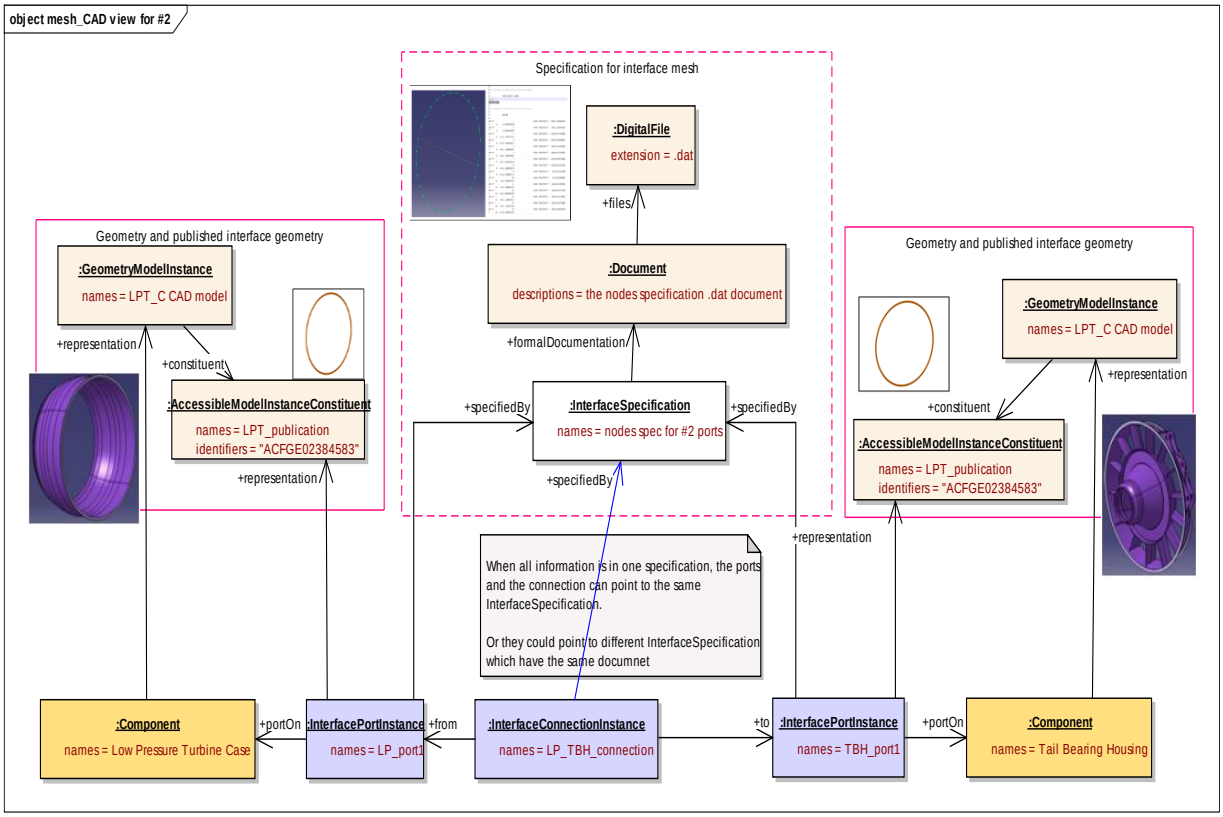


Figure 235: The interface definition via the system framework in the BDA BOM semantic

When the representation relationships are initially set up, it is an indication of what data needs to be created. For example, if a Mesh-ModelInstance is to be a representation of a Component, then the *ModelType* that the Instance "*isAnInstanceOf*", has documentation containing the definition of the expected mesh characteristics. The representations for ports can be used to indicate the type of boundary expected (e.g. a mesh, a node, a geometric surface or curve etc.), and also used to indicate the interface boundary conditions. For example, a typical "Product Integration" problem has the following steps:

- Step 1: an integrated model is created which is made up of simplified models of the elements being integrated;
- Step 2: From this, boundary conditions are extracted for the interfaces between the elements;
- Step 3: Each element is then analysed separately using more complex models;
- Step 4: New simplified models are then created;
- Step 5: The simplified models are then integrated back together into a single model for the next iteration.

Because all the interfaces are contained in the integrated model, it is a representation of all the Connections. The boundary conditions can be exposed either using *AccessibleModelInstanceConstituents*, or by creating new models containing just the boundary conditions that are derived from the integrated model. These become the representations for the *Ports*. Using boundary conditions in separate models has the advantage that separate element analysis steps only need the model of the boundary conditions, not the complete integrated model from which to access the constituents. This is illustrated in Figure 236 on the PPS IFEM generated for the product integration scenario presented in previous sections.

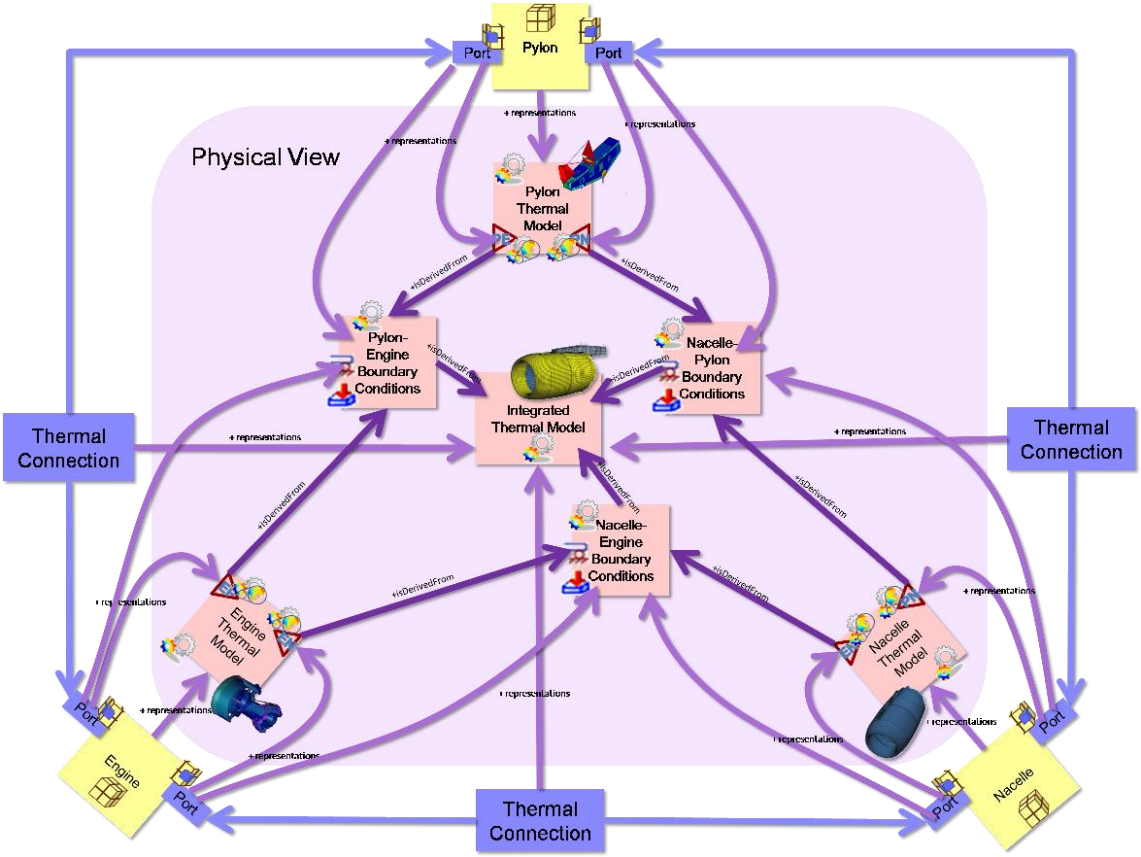


Figure 236: Representations of Interface elements in the Physical View

The above image also shows that the ports have representations which are *AccessibleModelInstanceConstituents* in the simplified models from step 4. This means that the integrated model of Step 5 is generated from the assembly of the models from Step 4 using the "recipe" defined by the port representations and connections between ports. For example the port on the engine that is connected to the pylon has a representation of a accessible constituent (e.g. publication) inside the Engine model. This publication can be matched up to the accessible constituent (publication) inside the Pylon model that is a representation of the port on the pylon that is connected to the engine. The rules for how to match them up will be contained inside the specifications for the ports (and connections).

At each iteration, new instances of the models are created in the Physical view, and these are representations of the same Components, Ports and Connections in the system view. So the Components, Ports and Connections will end up with many model representations, but those models will have *isEvolvedFrom* relationships making it possible to view their lifecycle.

13.3.2 Product Integration scenario example using BDA BOM Classes

The below diagrams (Figure 237, Figure 238 and Figure 239) show the same scenario but using the Business Object Model classes. The figures are laid out in the same way as the schematic figures above. The full example has both Front and Rear ports and connections, but only the fronts are shown for clarity.

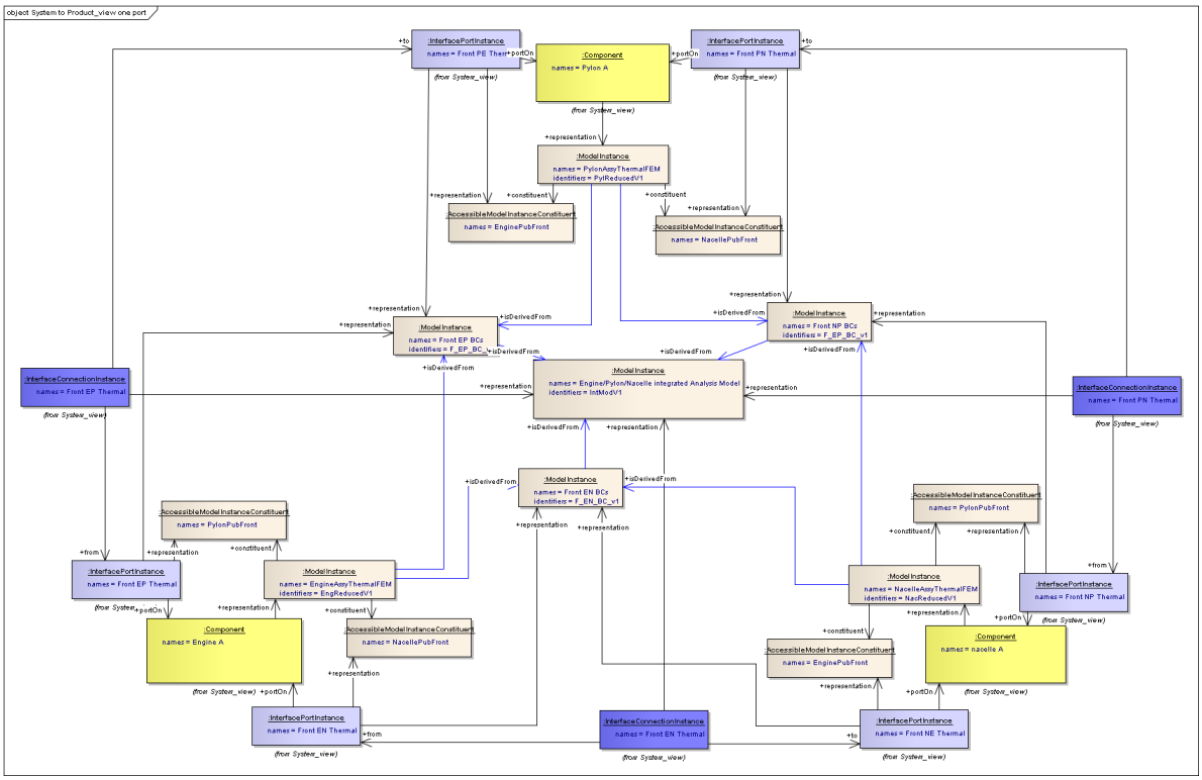


Figure 237: links between System and Product views for one iteration

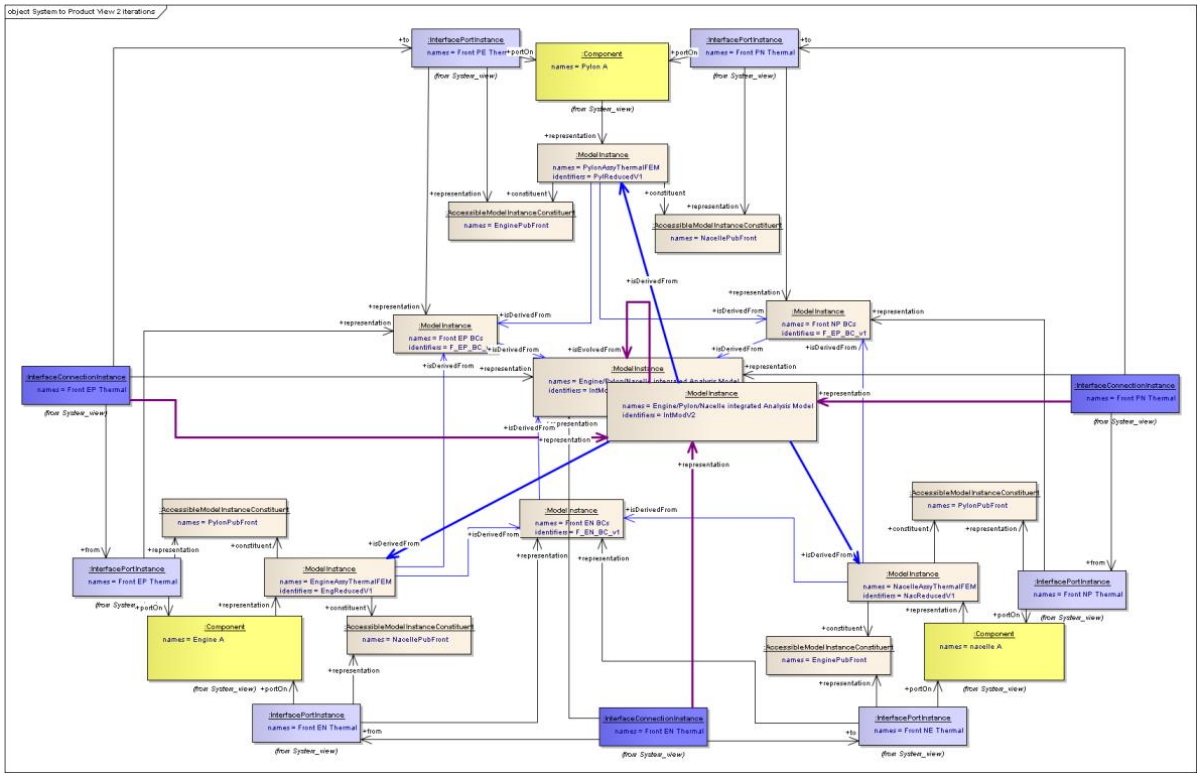


Figure 238: Adding Integrated model for second iteration - bold links

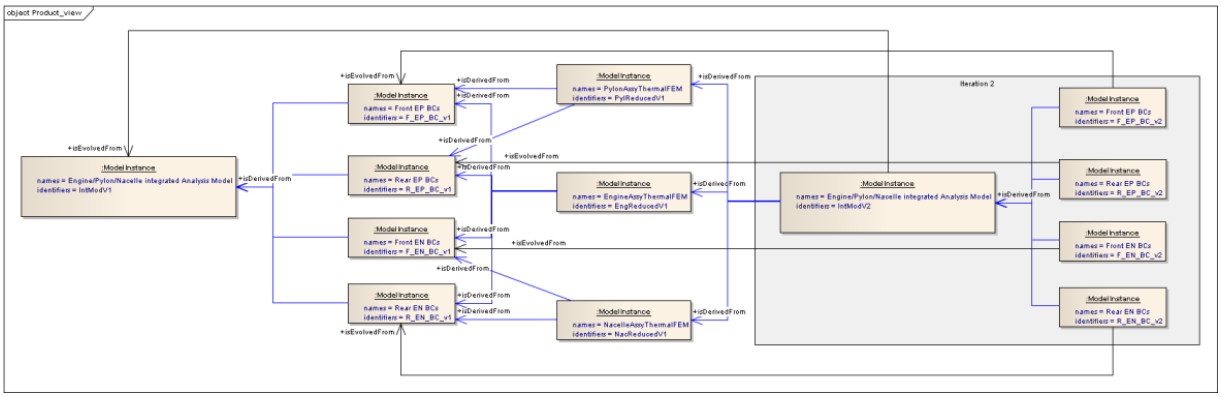


Figure 239: Product View subset: showing derived from and evolution relationships over two iterations

The "*isDerivedFrom*" relationships (in blue) are a simplification. The Business Object Model uses a *ModelInstanceIsDerivedFrom* class with additional relationships to other classes. Including these makes the diagram too complex so they have been replaced with a simple relationship instead. These relationships are managed by the Associative Model Network concept illustrated in Figure 240 below.

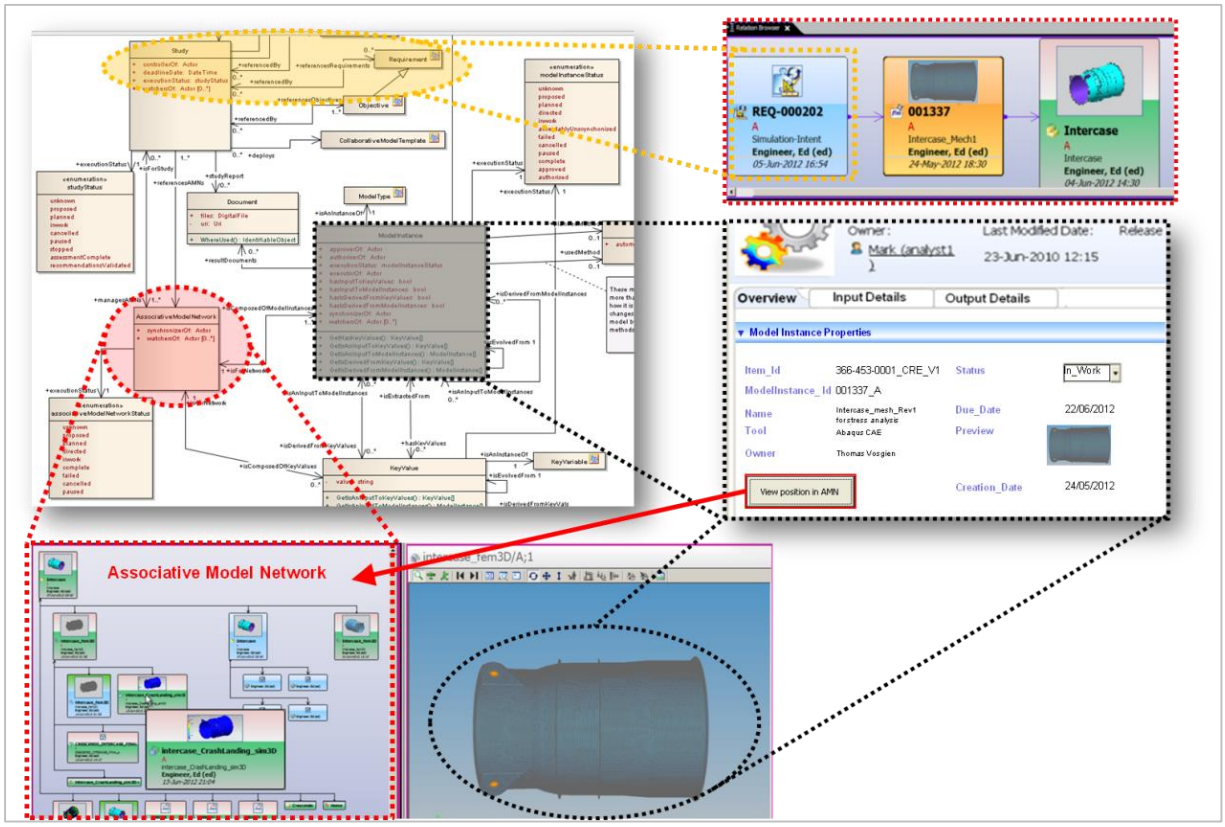


Figure 240: Illustration of the AMN concept and methodology – Meaning of the AMN BDA BOM object

13.4 Conclusion

The Product Integration scenario has permitted industrialists and software vendors to reach a higher maturity level about the understanding, the feasibility and implementability of the proposed DASIF concepts. The demonstrations have revealed current limits of commercial solutions that are detailed in section 14.1. This gap analysis will serve for future industrial specifications of PLM/SLM solutions for the ones who want to implement such concepts.

Concerning our contribution to the development and implementation of the BDA BOM, our scenario has permitted to highlight missing concepts and objects that cannot be handled by the BDA data model which only implements AP233 and AP239 protocols which were not developed for supporting collaborative simulation data management. Technical standards such as STEP-AP209 would have been needed to handle for instance the exchange of standardised FE models and results. No differentiation between standards for structuring the information model of the shared BDA repository and standards for exchanging data containers (models or results data files) has been made, generating hence ambiguities regarding the usage of such a common repository.

However, for the data exchange sequences of the Product Integration scenario, and thanks to the AP233 that covers SysML concepts and objects, we demonstrated that is was possible to exchange logical system architectures and related modelling requirements between two heterogeneous PLM/SLM systems (CATIA/ENVIA V6 and Teamcenter for Simulation 9).

Chapter 14: Results and validation

14.1 The Product Integration Scenario key results and gap analysis

Concerning the Product Integration demonstrators and related prototypes, five main proofs of concepts were developed:

- The **integrator / simulation architect dedicated environment** enabling to:
  - define and manage DDMU configurations and related logical architectures with a MBSE framework based on an object-oriented visual modelling language in order to prepare the creation of a fit-for purpose BMU;
  - integrate and link functional, structural, topological and behavioural specifications of system artefacts and especially components interactions and interfaces including multiple levels of hierarchy via ports and ports delegations;
  - automate the assembly of FEMs translating the complete BMU architecture into the appropriate pre-processing tool language.
- The **3D CAD and FEM interface templates** enabling to:
  - Create and exploit 3D-based interface specifications using the DMU models and making the links between CAD and FE topologies more explicit and traceable;
  - Capture and exploit both design and simulation intents of system artefacts (and in particular system interfaces) by accessing and capturing features present in CAD and CAE models and referencing them in the PDM/SDM system;
- **Semantic Data Mapping for interoperable SLM data exchange via the BDA BOM**: demonstrating an implementation of some of the BDA concepts and using web semantic technologies (RDF) to map the proprietary applications data models to the BDA BOM.
- Use **simulation intents** and modelling requirements to **create automatically fit-for-purpose meshes**.
- **Automated quality process** and visual reports to assess the compliance and the reliability of the provided FE models.

The demonstrations have revealed current limits of commercial solutions and the gap analysis regarding the initial technical objectives is detailed below. To make this gap analysis, we defined with other Crescendo industrial partners a certain number of assessment criteria based on the key innovations supposed to be brought by the BDA:

- **Multi-partner collaboration and engineering data exchange**: The ability to provide a flexible multi-partner collaboration based on standards and best practices ensuring security and trust.
- **Integrator / Simulation Architect dedicated environment** to manage product complexity in terms of:
  - Consistent organisation and configuration of multi-level and multi-domain system architectures representations;
  - Functional, structural, topological and behavioural definition and integration of system artefacts including multiple levels of hierarchy via ports and ports delegations optimizing the integration/assembly activities;
  - Dissemination of results to sub-systems and components levels and for weak multi-physic coupling.
- **Automation and "fit-for-purpose" FE modelling**: be able to trace and capture the modelling specifications/assumptions and to automate the related modelling routines/rules that lead to the creation of a finite element model for a specific study context and intent. The goal is to be

able, for a given study context and intent, to re-use the appropriate automated modelling procedures for new simulation scenarios involving different or updated data sets.

- **Re-usability** in order to capture engineering knowledge and avoid unnecessary rework by:
  - Ensuring the traceability of the design-analysis information chain for re-using suitable simulation data inputs and/or models
  - Providing libraries of re-usable and parameterised simulation and models templates enabling to re-use simulation workflows, product and model configurations, modelling requirements, automated modelling procedures, etc.
- **Models Quality**: ability to quickly check models reliability and ensure results accuracy without re-work

All the “Product Integration” scenario prototyped capabilities have been listed in following Table 12 and categorised according to the expected BDA innovations defined by the consortium. This table summarises the achievements of the demonstrator and the limitations faced for each capability implementation tested in commercial and proprietary applications.

BDA Technical Area / Innovation	Expected capabilities	Capability Id	PPI Achievements	PPI gap analysis
Multi-partner collaboration & Engineering data exchange	Standards based communications between heterogeneous PLM/SLM solutions	PPI-C1	Semantic Data Mapping for interoperable SLM data exchange via the BDA BOM: demonstrating an implementation of some of the BDA concepts and using web semantic technologies (RDF and OWL) to map the proprietary SLM applications data models to the BDA BOM.	The mapping need to be refined if the data models evolves: need for automatic routines enabling to update applications data models and to re-configure the mapping with the BDA BOM.
		PPI-C2	Enrichment of product data exchange standards for supporting design-analysis integration chains	By-pass limitation due to industries IT security in order to prove a complete interoperability in a collaborative environment
		PPI-C3	Standardized exchange of logical and behavioural DMU and BMU architectures with associated simulation intent and modelling requirements between two heterogeneous SLM solutions	For now, the BDA does not support the exchange/sharing of standardized and integrated CAD and CAE assembly models --> need to integrate technical standards such as STEP-AP209ed2.
	Interoperability between SLM and CAE pre-processing tools	PPI-C4	Ability to exploit BMU architectures defined in a SLM environment in an integrated CAE pre-processing tool	Further development between the description of simulation model from the "Integrator environment" and its fully exploitation by several pre-post tools.
	Security and Trust (Firewall, Encryption, Authentication)		not addressed	
	Flexibility: ability to easily plug any partner application to the BDA		not addressed	
Integrator / Simulation Architect dedicated environment	MBSE Architecture modelling for specifying "Fit for purpose" DDMU and BMU architectures	PPI-C5	Define and manage DDMU configurations and related logical architectures with a MBSE formalism in order to prepare the creation of a fit-for purpose BMU	Automate the re-organisation and filter of DMUs based on interfaces definition and on a neutral declarative language
	FBS integration of system artefact definition	PPI-C6	Integrate and link functional, structural, topological and behavioural specification of system artefacts and especially components interactions and interfaces including multiple levels of hierarchy via ports and ports delegations;	Facilitate the creation of the implementation links between functional, logical/structural and behavioural system artefacts and related design variables Facilitate the creation of the implementation links between CAX mating features and system objects (ports)
	Multi-level and multi-domain integration	PPI-C7	Manage data consistency between multi-level and multi-domain system representations	Coupling interdependent system views permitting multi-physics coupling
		PPI-C8	Navigate easily through the different system breakdown levels	OK
	CAD-CAE topological associativity	PPI-C9	Specify components and interfaces FE models using the DMU models and making the links between CAD and FE topologies more explicit and traceable;	Automate the extraction of CAD and CAE mating features and the creation of the topological associative links
		PPI-C10	3D CAD and FEM interface templates enabling to have 3D-based interface specifications and CAD-FE topological associations instead of using interface control documents that have no explicit links with DMUs or BMUs.	The FEM interface templates do not impose any topological constraints and can be inconsistent if one wants to re-use and apply them on unsuitable topological elements
	FE assembly automation	PPI-C11	Automate the assembly of FEMs translating the complete BMU architecture into the appropriate pre-processing tool language.	Capabilities only developed for Simulia-Abaqus CAE and TC4SIM-NX CAE. Need to have a standardized modelling language for describing these architectures and to generalise the implementation of these translators.
	Results dissemination	PPI-C12	Automated procedure to split results at interfaces and associate them to the appropriate interface ports on the system representation.	
		PPI-C13	Automatic interface results distribution at lower product breakdown levels via ports delegation	
Automation & Fit-for purpose FE modelling	Automatic CAD and CAE mating feature extraction	PPI-C14	Script automating CAD and CAE interfaces extraction in NX and NX-CAE	Capability and related scripts only implemented in NX-CAE
	CAD idealisation automation	PPI-C15	Script automating idealisation operations in NX-CAE based on simulation intent, topological rules and automatic recognition of topological elements types	
	Automatically create fit-for-purpose FE models	PPI-C16	Script automating meshing operations in NX-CAE based on simulation intent, modelling requirements and hypotheses and CAD-FE topological links	
Reusability	Re-use of simulation process and data inputs	PPI-C17	Re-use of simulation templates enabling faster data input collection and re-use of previous automated workflows	OK
		PPI-C18	Re-use of DDMU architectures and filters	OK
	Re-use of simulation model configurations	PPI-C19	Reuse architecture of the simulation model and associated data to modify data if any change	NOK - Be able to change any data in the system representation and update automatically the simulation model
	Traceability of the Design-Analysis information chain for re-using suitable simulation data inputs and/or models	PPI-C20	Associative model network (relation browser) enabling to capture and visualise the complete traceable design-analysis information chain	OK
	Re-use of automated modelling procedures	PPI-C21	Ability to reuse simulation intents and related modelling hypotheses and specifications to automatically create fit-for-purpose analysis models	OK
Models Quality	Ability to quickly check models reliability	PPI-C22	Automated procedure comparing modelling specifications (simulation intent) and provided models features	OK
		PPI-C23	3D visual report of FE models quality criteria	Capability only implemented in NX-CAE
	Ability to ensure results accuracy without re-work	PPI-C24	Automated workflow which ensure a comparison between two different models to associate an accuracy ratio to the model	No possibility to compare local area of the two meshes and ensure the consistency of the compared results. This issue is linked with the ability of post processing tools to coincide the two meshes. Moreover, this kind of procedure would be difficult to set-up in operational use for large assembly FE models because a fine mesh will be required for every component modelled in 2D or 3D.

Table 12: PPI capabilities - achievements and gap analysis

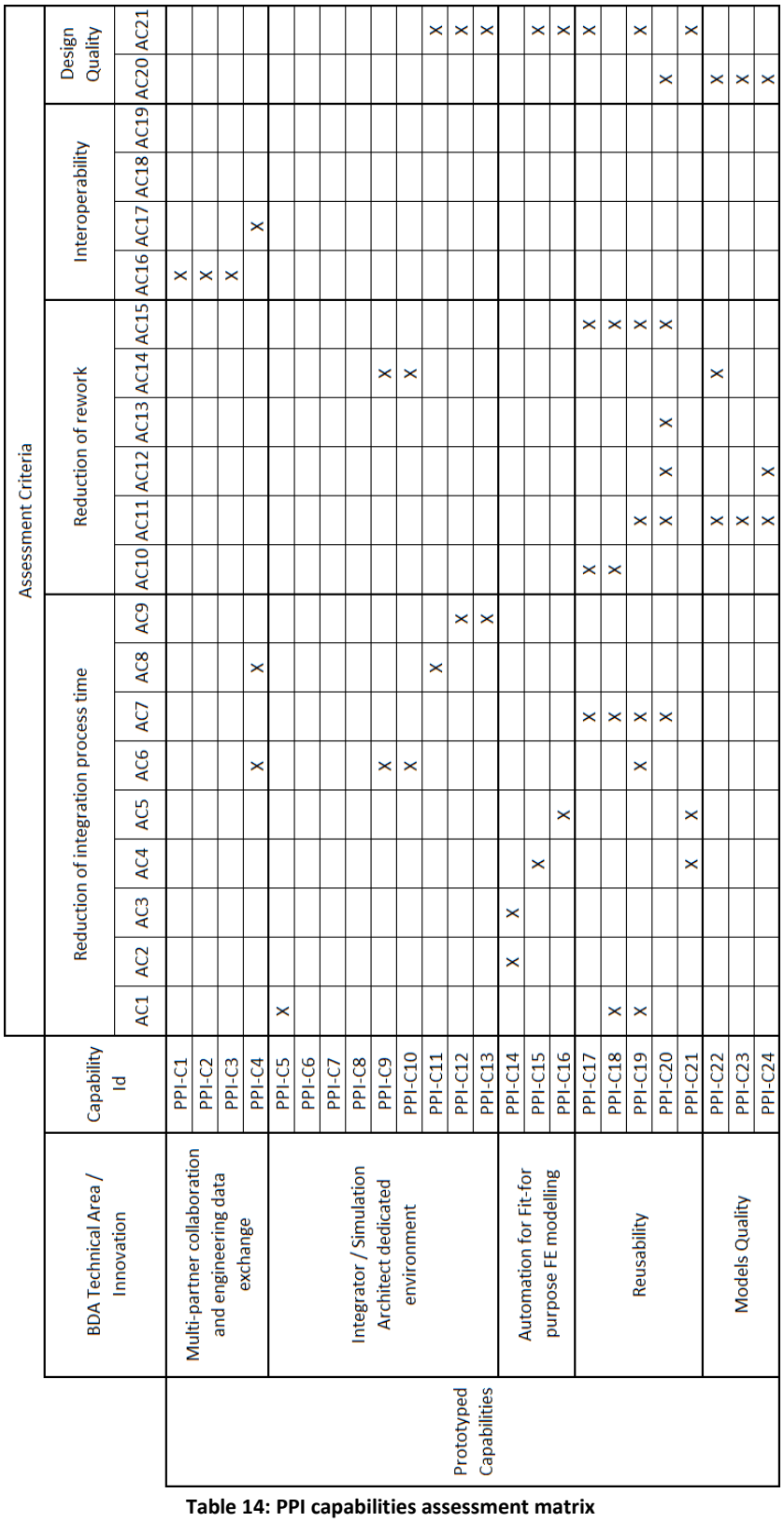
14.2 PPI capabilities assessment

The proposed and developed capabilities have not already been set up in an operational environment. Gains and benefits are then more difficult to assess. However, the main potential benefits of the proposed and developed demonstrator capabilities have been identified jointly with other industrial partners and can be synthesised as follows (see Table 13).

High-level assessment criteria	Criteria ID	Gain type	Assessment on PPI test-case (8 components developed by 8 different partners)		
			As-Is	To-Be	Approximate and relative benefits
Reduction of integration process time	AC1	Time for re-structuring a RDMu into a DDMU	not assessable		
	AC2	Time for capturing CAD mating features	1h	5min	92%
	AC3	Time for retrieving CAD and FE topological links	-	0 min	100%
	AC4	Time for idealising the DMU individual parts CAD models	3h	10 min	94%
	AC5	Time for creating individual parts meshes			
	AC6	Time for defining FE models interactions	2 h	0,5 h	25%
	AC7	Time to gather all necessary simulation data inputs	4 h	0,5 h	87,5%
	AC8	Time for assembling meshes	1 h	5 min	92%
	AC9	Time for disseminating results at interfaces	1 h	10 min	83%
Reduction of rework	AC10	Rework due to no possible re-use of DDMU architectures and filtering mechanisms	not assessable		
	AC11	Rework due to non-compliant models	not assessable		
	AC12	Rework due to inaccurate results	not assessable		
	AC13	Rework due to no synchronized product definitions	not assessable		
	AC14	Rework due to incompatible interfaces	not assessable		
	AC15	Rework due to no accessibility or traceability of re-usable simulation data	not assessable		
Interoperability	AC16	Cost due to the number of SLM-SLM translators to develop	56 translators	16 translators	-50% for 2 heterogeneous SLM 0% for 3 heterogeneous SLM 33% for 4 heterogeneous SLM 71% for 8 heterogeneous SLM ...
	AC17	Cost for developing SLM-CAE translators for exchanging and exploiting BMU architectures	Number of SLM-CAE transaction	Number of SLM + number of CAE tools	0% for 2 heterogeneous SLM and 2 heterogeneous CAE tool 33% for 3 heterogeneous SLM and 3 heterogeneous CAE tool ...
	AC18	Rework due to information loss in CAX data exchange	not assessable		
	AC19	Rework due to information loss in SLM data exchange	not assessable		
Design Quality	AC20	Reduction of the number of wrong simulation results interpretation	not assessable		
	AC21	Ability to simulate faster and hence to simulate more	not assessable		

Table 13: PPI capabilities - assessment criteria and related potential benefits

The “Verification” step of the initial planned research methodology introduced in Chapter 9, consisted in performing new Value Stream Mapping on the business processes studied at Snecma using the developed capabilities. However, since these capabilities could not be implemented and assessed in operational processes, we have decided to use a matrix-based approach to identify the potential benefits of the provided or specified capabilities. This matrix-based approach consists in crossing the capabilities coverage regarding the assessment criteria of Table 13. This enables to analyse and assess the impact of the proposed proof of concepts.



Prototyped Capabilities	BDA Technical Area / Innovation	Capability Id	Assessment Criteria: Reduction of integration process time AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	Reduction of rework AC10	AC11	AC12	AC13	AC14	AC15	Interoperability AC16	AC17	AC18	AC19	Design Quality AC20	AC21
	Multi-partner collaboration and engineering data exchange	PPI-C1																X					
		PPI-C2																X					
		PPI-C3																X					
		PPI-C4						X		X									X				
	Integrator / Simulation Architect dedicated environment	PPI-C5	X																				
		PPI-C6																					
		PPI-C7																					
		PPI-C8																					
		PPI-C9						X								X							
		PPI-C10						X								X							
		PPI-C11								X												X	
		PPI-C12									X											X	
		PPI-C13									X											X	
	Automation for Fit-for purpose FE modelling	PPI-C14		X	X																		
		PPI-C15				X																X	
		PPI-C16					X															X	
	Reusability	PPI-C17							X			X					X					X	
		PPI-C18	X						X			X											
		PPI-C19	X					X	X				X				X					X	
		PPI-C20							X				X	X	X		X					X	
		PPI-C21				X	X							X									X
	Models Quality	PPI-C22											X			X						X	
		PPI-C23											X									X	
		PPI-C24												X								X	

Table 14: PPI capabilities assessment matrix

14.3 DASIF concepts and capabilities assessment

Similarly to the assessment of the capabilities developed for the PPI demonstrator, we applied the same assessment method for analysing and assessing the impact of the proposed DASIF concepts capabilities. Conditions and capabilities - derived from the functional analysis of section , listed in section 10.2.3 and detailed for some of them in section 10.3 – are crossed in Table 15 with the same assessment criteria of Table 13.

DASIF addressed capabilities			Assessment Criteria																					
			Integration process time									Reduction of rework								Interoperability				Design Quality
DASIF technical area	Capability Id		AC1	AC2	AC3	AC4	AC5	AC6	AC7	AC8	AC9	AC10	AC11	AC12	AC13	AC14	AC15	AC16	AC17	AC18	AC19	AC20	AC21
DMUs consistency with product definition and other product representations	C0a														X							X	
	C0b														X							X	
	C0c														X							X	
Product configuration management process	C1		X																				
	C2																X						
	C3														X							X	
	C4														X		X					X	
	C5																						
DMU enrichment mechanisms	C6			X				X								X							
	C7			X				X								X							
	C8				X			X								X							
	C9				X			X															
	C10		X		X																		
DMU adaptation mechanisms	C11											X											
	C12		X																				
	C13											X											
	C14					X																	
links between DMUs and simulation data and meta-data	C15											X	X				X						
	C16																X					X	
	C17				X			X									X						
	C18		X	X		X						X	X				X						
interoperability and collaboration	C19																			X			
	C20																			X			
	C21																		X				
	C22																	X					

Table 15: DASIF concepts and capabilities assessment matrix

14.4 Conclusion

The assessment of the specified and/or prototyped capabilities – concerning whether DASIF whether the PPI demonstrator – will be used only when the implementation of these capabilities will be mature enough in existing commercial tools so that “Beta-test” procedures (assessment by the final users) can be performed and the efficiency of operational business processes will be measurable performing new VSMs. However, the combined usage of our gap analysis (performed and validated with other industrial partners) with the assessment matrices has permitted to validate the future development roadmap and future research work that we introduce in the next and final Chapter 15.

Chapter 15: Conclusions and future work

15.1 PhD Synthesis

15.1.1 Synthesis of addressed challenges and issues

The analysis of the industrial context and requirements introduced in Part I have highlighted some major industrial stakes and challenges related to the development of complex aeronautical systems and its supporting digital engineering technologies. Complex PDP is always subjected to risks, uncertainties, as well as difficulties to coordinate interdisciplinary design and simulation activities. All along an IPPS development life cycle, the tight coupling of several phenomena interacting together makes it a great integration challenge to find a global optimum for the entire system. The search for this optimal performance stresses the need to adopt an **integrated and simulation-based design approach.** The synchronisation and the reduction of time and knowledge gaps between multi-domain and multi-level interdependent simulations, performed by different partners using different tools, are major integration challenges. These gaps and uncertainties can be addressed by fastening the design-analysis iterations. The design-analysis iterations and integration challenges have been clarified with the help of the FBS framework. This has underlined the need to provide appropriate “definition view”; characterising the ideal content and organization of product data (corresponding to the appropriate behavioural structure) for a specific discipline and simulation context.

We then introduced and defined the concept of “Digital Integration Chains” as a process in which separately subsystems (or components) models are merged into one integrated product model enabling the analysts to predict the behaviour of the whole assembly and to validate that the integrated product delivers the expected system performances. In order to ensure the continuity of information within digital integration chains and between involved design teams and based on our analysis, we have identified three main challenges in complex product development:

- The **management of design and analysis data through an integrated reference framework**: this environment is required for a better integration and traceability between design and analysis data in view to provide needed “analysis product views” and to enhance re-use of relevant design artefacts (e.g. CAD models, physical properties or past analysis models) regarding the scope and objectives of the various performed analyses.
- **Interoperability between systems**: from our observation, we noticed that there was still a lack in integrating efficiently tools and the data exchanged through these tools. This lack of efficiency is currently a problem while attempting to exchange in a meaningful way between heterogeneous CAX and EDM systems; which explains the importance of implementing product data standards in such applications.
- **Consistency between data and models**: a consequence of interoperability constraints (even using standards) is the loss of data consistency between data managed in different domain-specific tools but also between tools of the same nature. The model consistency concerns the quality and the semantic of the data conveyed through these models. This loss of consistency causes the problem of data interpretation between multi-partner and multi-disciplinary co-designers.

Therefore, adequate collaborative design environments are necessary to ensure that partners and design teams can share or/and exchange the engineering data created all along the product development process. These collaborative design environments should enable partners involved in the ex-

tended enterprise to harmonize their design processes by exchanging the right data in the right context. The objective of defining such an integrated design environment is to provide and share an integrated and a common product definition for the project partners.

The Digital Mock-Up (DMU) – supported by Product Data Management (PDM) systems – be-came during the last decade a key federating environment to exchange/share a common 3D model-based product definition between design teams. It gives to designers and downstream users (analysts) an access to the geometry of the assembly. While enhancing 3D and 2D simulations in a collaborative and distributed design process, the DMU offers new perspectives for analysts to retrieve the appropriate CAD data inputs used for Finite Element Analysis (FEA); permitting hence to speed-up the simulation model generation. Another big DMU-related challenge for these companies to reach this objective and in view to extend the scope of PLM (by adding SLM functionalities), is to integrate behavioural simulation data and processes with the DMU. The PhD has been oriented for addressing this challenge: the ability for a DMU to represent a system from multiple viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail, fidelity and abstraction. However, current industrial DMUs suffer from several limitations that we have introduced and analysed in Chapter 4: the lack of flexibility in terms of content and structure, the lack of digital interface objects describing the relationships between its components and a lack of integration with simulation activities and data. This PhD especially underlines the DMU transformations required to provide adapted DMUs that can be used as direct input for FEA of large assemblies. These transformations must be consistent with the simulation context and objectives and lead to the concept of “Product View” applied to DMUs and to the concept of “Behavioural Mock-Up” (BMU). A product view defines the link between a product representation and the activity or process (performed at least by one stakeholder) that use or generate this representation as respectively input or output. The BMU is the equivalent of the DMU for simulation data and processes. Beyond the geometry, which is represented in the DMU, the so-called BMU should logically link all data and models that are required for simulating the physical behaviour and properties of a single component or an assembly of components.

15.1.2 Contribution summary

The key enabler for achieving the target of extending the concept of the established CAD-based DMU to the behavioural CAE-based BMU is to find a bi-directional interfacing concept between the BMU and its associated DMU. This concept is the kernel of the Design-Analysis Integration Framework (DASIF) proposed in this PhD. The objective was to develop the digital engineering capabilities and supporting data models enabling to manage “multi-level” and “multi-domain” logical and physical DMUs enriched with assembly information for providing to analysts and integrators the required data structures and inputs to automate the assembly of FE models constituting an appropriate and consistent integrated finite element model ready to be computed. The main approach consists in considering the DMU as a flexible product definition reference and the supporting PDM system as the main information source for designers and analysts collaborating within theses digital integration chains. The concepts of RDMU and DDMU have been introduced as well as the conditions and scenarios for building these DDMUs.

Regarding all these aspects, our proposal and contributions consist in the specification and development of the Design-Analysis System Integration Framework (DASIF). DASIF could be extended and to be used for any kind of design-simulation loops. We have focused on CAD-FEA loops. DASIF should be implemented within PLM/SLM environments and interoperate with both CAD-DMU and CAE-BMU

environments. DASIF combines configuration data management capabilities of PDM systems with system modelling concepts of MBSE and Simulation Data Management capabilities. We hence define our contribution according to the following developments:

- The development and implementation of MBSE concepts in product data management systems in order to:
  - Synchronise the DMU with the current product definition: using the DMU as the 3D-based referential product definition to derive other product representations used in downstream applications;
  - Provide an Architect/Integrator dedicated environments for design-analysis data integration enabling to:
    - Define and integrate (D)DMU and BMU architectures with an object-oriented modelling formalism enabling to:
      - encapsulate and re-use 3D-based interface specifications;
      - automate the assembly of FE models by translating the BMU architecture language in the appropriate implemented pre-processing tool language.
    - Consistently integrate Functional, Behavioural and Structural design parameters in a multi-view (multi-level and multi-domain) environment;
- The definition of the conceptual and logical multi-aspect product data models providing the required informational model to organise and manage the engineering artefacts used within the DASIF framework. This data models have several objectives:
  - To be enriched with required attributes and operations/methods according to the specific business requirements of the environment where it has to be implemented;
  - To be implemented in a relational database and its client applications constituting the product and simulation data management systems supporting DASIF;
  - To specify the data structure of exchanged CAD and CAE containers (digital data files);
  - To contribute to the enrichment of the BDA object model with the proposed concepts and services so that they can potentially be implemented within the BDA hub platform in order to support collaborative data exchange within CAD-CAE digital integration chains.
- The development of the interoperability between engineering applications supporting digital integration chains. This has been addressed by:
  - Combining and adapting product data models mainly based on existing standards for product data exchange;
  - Defining a modular and multi-layered “standardised” data model enabling to dissociate but also integrate configuration management data, system definition data (functional, structural and behavioural engineering artefacts) and topological data. Finally an applicative layer has been added to manage the required import/export functions enabling to transform DASIF language constructions into the appropriate languages for CAD and CAE authoring tools.
  - Using semantic data mapping and the technology of web semantics for heterogeneous PLM/SLM data exchange; with the BDA BOM as neutral data model.

This PhD work has been carried out within a European research program: the CRESCENDO project which aims at delivering the Behavioural Digital Aircraft (BDA). Along the CRESCENDO project, partners have proposed and assessed innovations that improve the way of collaboratively developing and simulating aeronautical products by developing both modelling/simulation and SDM capabilities in order to:

- Reduce lead time in performing modelling and analysis activities;
- Provide accurate and robust simulation for product behavioural analyses;
- Enhance the cross-domain interaction by using multi-physics and distributed simulation;
- Improve the quality of modelling to limit re-work and computational issues.

The work package in which this PhD work was integrated, has studied different aspects of simulation in detailed design phase. Within this work-package, the Product Integration Scenario and related methodology have been defined to handle digital integration chains and to provide a test case scenario for assessing DASIF concepts. These latter have been used to specify and develop a prototype of an "Integrator Dedicated Environment" implemented in commercial PLM/SLM applications (CATIA/SIMULIA V6 and Teamcenter for Simulation 9). These prototypes have allowed assessing the current commercial tools maturity regarding these concepts and have a feedback regarding the feasibility of their implementation. Finally the conceptual data model of DASIF has also provided inputs to define the Behavioural Digital Aircraft Business Object Model: the standardized data model of the BDA platform enabling interoperability between heterogeneous PLM/SLM systems and to which existing local design environments and new services to be developed could plug. The product Integration scenario has permitted industrialists and software vendors to reach a higher maturity level about the understanding, the feasibility and implementability of the proposed DASIF concepts. The main achievements have been:

- The automated assembly of FE models by using the logical DDMU and BMU system architectures defined in the system modelling framework of the integrator dedicated environment;
- Full traceability of the CAD-CAE data chain;
- Re-use or automated generation of fit-for-purpose meshes based on simulation intent.

These demonstrations have revealed current limits of commercial solutions that are detailed in section 14.1. This gap analysis will serve for future industrial specifications of PLM/SLM solutions for the ones who want to implement such concepts. The developed prototypes would also need further capabilities developments for:

- Improving the automation of the CAD and CAE features extraction and the automation of creating the links between CAX models' features and system objects.
- Better integration between CAD and CAE data within the implemented solution for traceability purpose.
- Application and demonstrations of the proposed approach and concepts in other disciplines (e.g. Aerothermal analysis and management of fluid interfaces)
- Coupling interdependent system views permitting multi-physics coupling (e.g. thermo-mechanical analysis and thermo-mechanical interfaces management).

Concerning our contribution to the development and implementation of the BDA BOM, our scenario has highlighted missing concepts and objects that cannot be handled by the BDA data model which only implements AP233 and AP239 step protocols which were not developed for supporting collaborative simulation data management. Technical standards such as STEP-AP209 would have been necessary to handle for instance the exchange of standardised FE models and results. No differentiation between standards for structuring the information model of the shared BDA repository and standards for exchanging data containers (models or results data files) has been made. For the data exchange sequence of the Product integration scenario, and thanks to the AP233 that covers SysML concepts and objects, we demonstrated that it was possible to exchange logical system architectures and related modelling requirements between two heterogeneous PLM/SLM systems (CATIA/ENOVIA V6

and Teamcenter for Simulation 9). The demonstrators have also highlighted some limitations regarding standard implementation and usage:

- functional scope and information coverage of the standards regarding specific business needs
- no way of assessing the quality/conformity of standards implementations in PLM solutions

Finally, in this PhD we adopted a MBSE approach to introduce a systems engineering dimension in current PLM/SLM systems. Using an extended SysML notation and based on an innovative model-based and feature-based interface data model, the proposed approach provides a robust interface modelling capability to designers and integrators. The proposed "integrator dedicated framework" also allows to have a multi-level (hierarchical), multi-physics and multi-domain system and interfaces characterization. This framework and related new digital capabilities aim at improving the lead time for setting-up and simulating an integrated product. The approach consists mainly by enriching the engineering knowledge contained in CAX models and enabling to adapt DMU structure and content in order to reduce time and rework in design/analysis modelling and data processing (acquiring, structuring, analyzing, verification, etc.) activities. A barrier preventing to achieve these objectives is that CAD and PDM systems used as demonstration platforms are supported by data models that are not currently adapted to make explicit links between CAD and CAE data. Moreover they do not provide neither consistent multiple views of the product regarding the simulation objectives. Indeed, when using an integrated system model, the modelling language must explicitly allow for information to be shared in different views. Implementing such a framework in **PDM and PLM systems requires a general transformation towards better standardization, data consistency, and concentration on a few robust sources. The MBSE approach must be part of this new PLM strategy**.

15.2 Future work

One lesson learnt from this PhD is the understanding and vision of the PLM and Systems Engineering approaches. PLM is not a tool but a strategy for managing complex System of Systems in dynamic manufacturing networks. PLM is itself a system of systems in which different systems of interest need to be considered. According to ISO 15288 [ISO, 2008], the system of interest, i.e. the manufactured product, and the supporting systems, i.e. system for designing, manufacturing, operating and supporting the product, are distinguished. For each of them, all of the phases of the lifecycle are to be considered in order to ensure adequacy between industrial processes and enterprises' capabilities. Within a dynamic manufacturing network, all these different systems evolve during the product life cycle – including the organisations and the software applications – making the configuration management of these systems even more complex to handle and leading to other extensions of the PLM approach such as Application Lifecycle Management (ALM).

Therefore, some overlapping exists between PLM and Systems Engineering. The scope of application of PLM is larger than the one covered by Systems Engineering processes and can be applied being Systems Engineering processes independent. PLM is also more concerned by the information system and by the technical applications, while Systems Engineering is more concerned by engineering methods and processes. Finally, both Systems Engineering and PLM are concerned by interoperability. While PLM is concerned by data exchange, sharing and long term archiving, Systems Engineering is concerned not only by possible interaction between systems and by automated reconfiguration of systems of systems but also by enhancing communication and hence interoperability between multi-disciplinary design teams. Moreover, Systems Engineering also focuses on the adaptation of the overall system in

order to achieve the targeted objectives and on the way the different sub-systems of a system of systems have to be aggregated dynamically and to interact easily. This PhD has focused on the system to develop (aeronautical manufactured products) and the related systems (EDM and CAX systems) for designing and simulating the manufactured product.

This PhD has raised multiple challenges concerning the ability to develop a "central reference" for design, integration and simulation activities to make the right data and information available at the right time and for the right actor to handle efficiently digital integration chains.

We are convinced that a combination of both macroscopic approaches (defining the necessary data structure to be implemented in PDM applications to manage design-analysis data integration issues as well as to manage multiple-product data views and the change propagation procedures across these views) and microscopic approaches (defining the required mechanisms for simulation-driven structural and shapes DMU adaptations) is required. This combination must be considered as a crucial step toward the integration of MBSE concepts and methods in the PLM strategy required for the consistency of product data along the product lifecycle and for the interoperability of engineering applications in dynamic manufacturing networks.

Then still open research outlines for future work have been identified to take the plunge of this technological leap:

- In the area of Design-Analysis (CAD-CAE) integration:
  - Our contribution should permit to further develop and implement DDMU and BMU concepts for demonstrating their operational usage and efficiency on realistic industrial test-cases (thousands of parts) and for other kind of simulations and application domains. As stated above the management of consistent DDMUs will be operational when we will be able to drive and automate the DMU transformations operators (shape and structural transformations) within PDM and SDM systems using object-oriented modelling techniques and languages such as graph of relations based on graph theories or system modelling frameworks;
  - The implementation of Simulation Data Management and adapted configuration management capabilities as well as a full integration with PDM and CAX systems are still ongoing challenges for researchers and software vendors. Some of these capabilities still require efforts to be implemented like the management of BMUs configurations, their links with associated DDMUs and their application context (domain of validity). We propose a data model supporting these configuration management capabilities for FEA (to be able to manage the effectivities of the various FE representations (0D, 1D, 2D, 3D) in a BMU) but they could not be implemented during this PhD;
  - Simulation Lifecycle Management is dedicated to the management and capitalisation of the intellectual property related to simulation methods, data, and processes. Therefore a more long-term perspective is to develop and implement Knowledge-Based Engineering and expert systems in SDM and CAE tools in order to:
    - Capture the tacit knowledge of designers, analysts and other experts (e.g. the justification of modelling choices and specifications regarding the simulation context);
    - Re-produce the cognitive mechanisms developed by the experts and be able to re-use and automate modelling and simulation procedures for new simulation scenarios.

- The extension of the Downstream DMUs (DDMU) concept for other downstream applications and application domains. Among the possible downstream applications we can quote DDMUs for manufacturing and DDMUs for support and maintenance.
- In the area of PLM/SLM interoperability and standards:
  - Extension of standardised object-oriented modelling languages for specifying
    - Structural and behavioural architectures: enabling system models to better integrate geometric and behavioural models and parameters;
    - Simulation models and models interfaces: the need for a standardised language for defining and exchanging these specifications between multi-level and multi-domain design teams is crucial and represents a short-term perspective for industrialists.
  - Reduce the number of redundant units of functionalities within the different STEP application protocols. This is the aim of the AP242 standard merging AP214 developed for the automotive industry and AP203 developed for the aeronautics industry. For handling design-analysis interoperability issues, the next step consists in developing an application protocol merging AP242, AP233 and AP209 with dedicated units of functionalities for structuring and standardising data models of PDM/SDM shared repositories and for standardising exchanged PDM/SDM and CAD/CAE data containers.
  - Reduce the time for developing and implementing STEP application protocols: this could be addressed by developing an experimental and open platform for standards implementations assessment. This platform should allow to industrialists to better specify their data exchange scenarios with a model-driven approach (MDA) in order to simulate these scenarios by plugging their applications (implementing the standards) on the platform. This kind of platform should integrate technologies such as virtualisation clusters, workflow engines, enterprise service bus (ESB), application and data integration servers, large triple stores and a set of referential standards-oriented quality checking tools.
  - Create a community integrating the stakeholders of the PLM interoperability eco-system (industrial associations (AIA, ASD-SSG, GALIA, etc.), software vendors, research institutes and universities, standardisation organisms, etc.) to share the referential implementations of standards, quality checkers, industrial test-cases and best-practices with the finality to increase the maturity of the eco-system and speed-up the standards development and implementation process.
  - The development of SLM Hubs (further BDA developments for the European aeronautical industry): once required SDM capabilities and PLM standards implementations will have reached a sufficient maturity level, the next step will be to develop PLM/SLM collaborative hubs. The objective of the Crescendo project – developing the BDA platform – was may be too ambitious at the time the project was launched. However, it has enabled industrial partners and software vendors to reach a higher maturity level in the development and implementation of modelling and simulation capabilities for collaborative simulation-based design processes in the extended enterprise.

Acknowledgements

The research leading to these results has received funding from the European Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 234344 (www.crescendo.fp7.eu/).

REFERENCES

3SL (2013) Cradle from 3SL.

AFIS, G. D. T. I. S. (2009) Découvrir et comprendre l'Ingénierie Système. Association Française d'Ingénierie Système.

AGOSTINHO, C., CORREIA, F. & JARDIM-GONCALVES, R. (2010) Interoperability of complex business networks by language independent information models. *New World Situation: New Directions in Concurrent Engineering.* Springer.

AHMAD, N. (2010) Supporting the management of the engineering change management process through a cross-domain traceability model. PhD Thesis, University of Cambridge.

ALEMANNI, M., DESTEFANIS, F. & VEZZETTI, E. (2011) Model-based definition design in the product lifecycle management scenario. *The International Journal of Advanced Manufacturing Technology,* 52**,** 1-14.

ANDERSSON, K. (1999) A design process model for behavior simulations of complex products. IN ASME (Ed.) *DETC 1999, Computers in Engineering Conference.* Las Vegas Nevada, USA

ANDERSSON, K. & SELLGREN, U. (1998) MOSAIC - Integrated modeling and simulation of physical behavior of complex systems. *Proceedings of NordDesign 98* Stockholm, Sweden.

ANDÚJAR, C., BRUNET, P. & AYALA, D. (2002) Topology-reducing surface simplification using a discrete solid representation. *ACM Transactions on Graphics (TOG),* 21**,** 88-105.

ANSI, A. N. S. I., SYSTEMS ENGINEERING COMMITTEE. (1999) Standard: ANSI/EIA-632 - Processes for Engineering a System.

ARABSHAHI, S., BARTON, D. C. & SHAW, N. K. (1991) Towards integrated design and analysis. *Finite Elements in Analysis and Design,* 9**,** 271-293.

ARABSHAHI, S., BARTON, D. C. & SHAW, N. K. (1993) Steps towards CAD-FEA integration. *Engineering with Computers,* 9**,** 17-26.

ARMSTRONG, C. G. (1994) Modeling Requirements for Finite-Element Analysis. *Computer-Aided Design,* Volume 26, Number 7**,** pp. 573-578.

ARMSTRONG, C. G., DONAGHY, R. J. & BRIDGETT, S. J. (1996) Derivation of appropriate Idealizations in Finite Element Modeling. *Third International Conference on Computational Structures Technology.* Budapest, Hungary.

BADIN, J. (2011) Ingénierie hautement productive et collaborative à base de connaissances métier : vers une méthodologie et un méta-modèle de gestion des connaissances en configurations. *Thèse de Doctorat,* Université de Technologie de Belfort-Montbéliard (UTBM).

BADIN, J., CHAMORET, D., GOMES, S. & MONTICOLO, D. (2011) Knowledge configuration management for product design and numerical simulation. *Proceedings of the 18th International Conference on Engineering Design (ICED11), Vol. 6.* Copenhague, Danemark.

BAJAJ, M. (2008) Knowledge Composition Methodology for Effective Analysis Problem Formulation in Simulation-based Design. *Engineering Information Systems Lab* Atlanta, USA, Georgia Institute of Technology

BAJAJ, M., PEAK, R. S. & PAREDIS, C. J. (2007) Knowledge Composition for Efficient Analysis Problem Formulation: Part 2 Approach and Analysis Meta-Model. ASME.

BAJAJ, M., ZWEMER, D., PEAK, R., PHUNG, A., SCOTT, A. & WILSON, M. (2011) Satellites to Supply Chains, Energy to Finance - SLIM for Model-Based Systems Engineering. *INCOSE International Symposium.*

BALLU, A., FALGARONE, H., CHEVASSUS, N. & MATHIEU, L. (2006) A new Design Method based on Functions and Tolerance Specifications for Product Modelling. *CIRP Annals - Manufacturing Technology,* 55**,** 139-142.

BAUCH, C. (2004) Lean product development: making waste transparent. Massachusetts Institute of Technology and Technical University of Munich.

BELAZIZ, M., BOURAS, A. & BRUN, J. M. (2000) Morphological analysis for product design. *Computer-Aided Design,* 32**,** 377-388.

BELLENGER, E., BENHAFID, Y. & TROUSSIER, N. (2008) Framework for controlled cost and quality of assumptions in finite element analysis. *Finite Elements in Analysis and Design,* 45**,** 25-36.

BENCHIMOL, G. (1993) *L'entreprise étendue (Coll. Systèmes d'information),* Paris.

BERGENTHAL, J. (2011) Final Report - Model Based Engineering (MBE) Subcommittee. NDIA Systems Engineering Division - M&S Committee.

BNAE (2003) Guide pour la mise en oeuvre des principes de la gestion de la configuration. *Recommandations Générales.* Bureau de normalisation de l'aéronautique et de l'espace, Issy les Moulineaux, France.

BOOCH, G. (2006) *Object Oriented Analysis & Design with Application*, Pearson Education India.

BOOCH, G., JACOBSON, I. & RUMBAUGH, J. (2000) OMG unified modeling language specification. *Object Management Group ed: Object Management Group***,** 1034.

BOUSSUGE, F., LÉON, J.-C., HAHMANN, S. & FINE, L. (2012) An analysis of DMU transformation requirements for structural assembly simulations. *The Eighth International Conference on Engineering Computational Technology, 4-7 September 2012, Dubrovnik, Croatia.*

BRONSVOORT, W. F. & NOORT, A. (2004) Multiple-view feature modelling for integral product development. *Computer-Aided Design,* 36**,** 929-946.

CAPRA, F. (1997) *The web of life: A new scientific understanding of living systems*, Anchor.

CHAMBOLLE, F. (1999) Un modèle produit piloté par les processus d'élaboration: application au secteur automobile dans l'environnement STEP. *Laboratoire Génie Industriel.* Chatenay-Malabry, FRANCE, Ecole centrale des arts et manufactures.

CHARLES, S. (2005) Gestion Intégrée des données CAO et EF – Contribution à la liaison entre conception mécanique et calcul de structure. *Conception Mécanique et Ingénierie Simultanée.* Thèse de doctorat, Université de Technologie de Troyes.

CHARLES, S., DUCELLIER, G. & EYNARD, B. (2006) CAD and FEA Integration in a Simulation Data Management Environment based on a knowledge-based system. Proceedings of TMCE 2006, April 18–22, 2006, Ljubljana, Slovenia.

CHARLES, S., EYNARD, B., BARTHOLOMEW, P. & PALECZNY, C. (2005) Standardization of the finite element analysis data-exchange in aeronautics concurrent engineering. *Journal of Computing and Information Science in Engineering,* 5**,** 63-66.

CHASE, J. P. (2001) Value creation in the product development process. Massachusetts Institute of Technology.

CHEN, Y.-M. & JAN, Y.-D. (2000) Enabling allied concurrent engineering through distributed engineering information management. *Robotics and Computer-Integrated Manufacturing,* 16**,** 9-27.

CHONG, C., SENTHIL KUMAR, A. & LEE, K. (2004) Automatic solid decomposition and reduction for non-manifold geometric model generation. *Computer-Aided Design,* 36**,** 1357-1369.

CHOUADRIA, R. & VERON, P. (2006) Identifying and re-meshing contact interfaces in a polyhedral assembly for digital mock-up. *Engineering with Computers,* 22**,** 47-58.

CLARK, B. W., HANKS, B. W. & ERNST, C. D. (2008) Conformal assembly meshing with tolerant imprinting. *Proceedings of the 17th International Meshing Roundtable.* Springer.

CLARK, K. B. & FUJIMOTO, T. (1991) *Product development performance: Strategy, organization, and management in the world auto industry*, Harvard Business Press.

CLARKSON, P. J., SIMONS, C. & ECKERT, C. (2004) Predicting Change Propagation in Complex Design. *Journal of Mechanical Design,* 126**,** 788-797.

CLIFT, T. B. & VANDENBOSCH, M. B. (1999) Project Complexity and Efforts to Reduce Product Development Cycle Time. *Journal of Business Research,* 45**,** 187-198.

CRESCENDO, C. (2010) Crescendo FP7 Europeean Project.

CRESCENDO, C. (2013) Crescendo FP7 Europeean Project.

CROW, K. (2002) Configuration management and engineering change control.

D'ADDERIO, L. (2001) Crafting the virtual prototype: how firms integrate knowledge and capabilities across organisational boundaries. *Research Policy,* 30**,** 1409-1424.

DAVIS, J. M., KEYS, L. K., CHEN, I. J. & PETERSEN, P. L. (2004) Collaborative product development in an R&D environment. *National Aeronautics and Space Administration, E-14440*.

DE LA BRETESCHE, B. (2000) *La Méthode APTE: analyse de la valeur, analyse fonctionnelle*, Ed. Pétrelle.

DE MARTINO, T., FALCIDIENO, B. & HAßINGER, S. (1998) Design and engineering process integration through a multiple view intermediate modeller in a distributed object-oriented system environment. *Computer-Aided Design,* 30**,** 437-452.

DEFOORT, S., BALESDENT, M., KLOTZ, P., SCHMOLLGRUBER, P., MORIO, J., HERMETZ, J., BLONDEAU, C., CARRIER, G. & BÉREND, N. (2012) Multidisciplinary Aerospace System Design: Principles, Issues and Onera Experience. *Journal of Aerospace Lab*.

DEMOLY, F. (2010) Conception intégrée et gestion d'informations techniques : application à l'ingénierie du produit et de sa séquence d'assemblage. Thèse de doctorat, Université de Technologie de Belfort-Montbéliard (UTBM), France.

DEMOLY, F., GOMES, S., EYNARD, B. & RIVEST, L. (2010a) PLM-based approach for Assembly Process Engineering. *International Journal of Manufacturing Research,* 5**,** 413-428.

DEMOLY, F., MONTICOLO, D., EYNARD, B. T., RIVEST, L. & GOMES, S. (2010b) Multiple viewpoint modelling framework enabling integrated product-process design. *International Journal on Interactive Design and Manufacturing (IJIDeM),* 4**,** 269-280.

DEMOLY, F., TOUSSAINT, L., EYNARD, B., KIRITSIS, D. & GOMES, S. (2011a) Geometric skeleton computation enabling concurrent product engineering and assembly sequence planning. *Computer-Aided Design,* 43**,** 1654-1673.

DEMOLY, F., TROUSSIER, N., EYNARD, B., FALGARONE, H., FRICERO, B. & GOMES, S. (2011b) Proactive assembly oriented design approach based on the deployment of functional requirements. *Journal of Computing and Information Science in Engineering,* 11.

DEMOLY, F., YAN, X.-T., EYNARD, B., GOMES, S. & KIRITSIS, D. (2011c) Integrated product relationships management: a model to enable concurrent product design and assembly sequence planning. *Journal of Engineering Design,* 23**,** 544-561.

DEMOLY, F., YAN, X.-T., EYNARD, B., RIVEST, L. & GOMES, S. (2011d) An assembly oriented design framework for product structure engineering and assembly sequence planning. *Robotics and Computer-Integrated Manufacturing,* 27**,** 33-46.

DIAZ-CALDERON, A. (2000) A composable simulation environment to support the design of mechatronic systems. Carnegie Mellon University.

DIAZ-CALDERON, A., PAREDIS, C. J. & KHOSLA, P. K. (2000) Architectures and languages for model building and reuse: organization and selection of reconfigurable models. *Proceedings of the 32nd conference on Winter simulation.* Society for Computer Simulation International.

DO, N., CHOI, I. J. & SONG, M. (2008) Propagation of engineering changes to multiple product data views using history of product structure changes. *International Journal of Computer Integrated Manufacturing,* 21**,** 19-32.

DOD, M.-H. (2001) 61A (SE) Military Handbook Configuration Management Guidance. *DoD, US*.

DOLEZAL, W. R. (2008) Success Factors for Digital Mock-ups (DMU) in complex Aerospace Product Development. *Technische Universität München, Genehmigten Dissertation, Munich, Germany*.

DRIEUX, G. (2006) De la maquette numérique produit vers ses applications aval: propositions de modèles et procédés associés. *Thèse de doctorat, Laboratoire Sols, Solides, Structures de Grenoble.* Grenoble, INPG.

DRIEUX, G., LÉON, J. C., GUILLAUME, F., CHEVASSUS, N., FINE, L. & POULAT, A. (2007) Interfacing product views through a mixed shape representation. Part 2: Model processing description. *International Journal on Interactive Design and Manufacturing (IJIDeM),* 1**,** 67-83.

DS, D. S. (2010) PLM (Product Lifecyle Management) *PLM Glossary.*

DÜSING, C. (2000) Towards the emerging Systems Engineering data exchange standard – AP233. *ProSTEP Science Days Conference Paper.*

ECKARD, C. (2000) Advantages and disadvantages of fem analysis in an early state of the design process. *Proc. 2nd worldwide automotive conf., MSC software corp., USA.*

ECKERT, C. M., KELLER, R., EARL, C. & CLARKSON, P. J. (2006) Supporting change processes in design: Complexity, prediction and reliability. *Reliability Engineering & System Safety,* 91**,** 1521-1534.

ESPOSITO, E. (2004) Strategic alliances and internationalisation in the aircraft manufacturing industry. *Technological Forecasting and Social Change,* 71**,** 443-468.

ESTEFAN, J. A. (2007) Survey of model-based systems engineering (MBSE) methodologies. *Incose MBSE Focus Group,* 25.

EUROSTEP (2013) PLCSlib and DEX repository.

EYNARD, B. & YAN, X.-T. (2008) "Collaborative Product Development''. *Concurrent Engineering,* 16**,** 5-7.

EYNARD, B. T., GALLET, T., ROUCOULES, L. & DUCELLIER, G. (2006) PDM system implementation based on UML. *Mathematics and Computers in Simulation,* 70**,** 330-342.

FALGARONE, H. & CHEVASSUS, N. (2006) Structural and Functional Analysis for Assemblies. *Advances in Design.* Springer London.

FENG, G., CUI, D., WANG, C. & YU, J. (2009) Integrated data management in complex product collaborative design. *Computers in Industry,* 60**,** 48-63.

FENVES, S., FOUFOU, S., BOCK, C. & SRIRAM, R. D. (2007) CPM: a core model for product data. *Manufacturing Systems, Integration, Division, National Institute of Standards and Technology, Gaithersburg, USA Available at: http://www.mel.nist.gov/msidstaff/sriram/. Accessed on: March 10th*.

FENVES, S. J. (2002) *Core Product Model for Representing Design Information*, US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

FENVES, S. J., CHOI, Y., GURUMOORTHY, B., MOCKO, G. & SRIRAM, R. D. (2003) Master Product Model for the Support of Tighter Integration of Spatial and Functional Design. Gaithersburg, USA., US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

FERRANDES, R., MARIN, P., LEON, J.-C. & GIANNINI, F. (2009) A posteriori evaluation of simplification details for finite element model preparation. *Computers & Structures,* 87**,** 73-80.

FINE, L. (2001) Processus et méthodes d'adaptation et d'idéalisation de modèles dédiés à l'analyse de structures mécaniques. Thèse de doctorat, Institut nationale polytechnique de Grenoble.

FISCHER, M. & SACHERS, M. (2002) ISO10303-214 CC8 Recommended Practices, Version 1.2: Specification and Configuration / Product Coding / Documents and Transformation Matrices.

FOUCAULT, G. & LÉON, J.-C. (2010) Enriching assembly CAD models with functional and mechanical informations to ease CAE. *Proceedingfs of the ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2010 August 15-18, 2010, Montréal, Canada.*

FOUCAULT, G., SHAHWAN, A., LÉON, J.-C. & FINE, L. (2011) What is the content of a DMU? Analysis and proposal of improvements. *Actes du 12ème Colloque National AIP PRIMECA: Produits, Procédés et Systèmes Industriels - intégration Réel-Virtuel.* Le Mont Dore, France.

FRIEDENTHAL, S., GRIEGO, R. & SAMPSON, M. (2007) INCOSE Model Based Systems Engineering (MBSE) Initiative. San Diego, INCOSE.

FRIEDENTHAL, S., MOORE, A. & STEINER, R. (2011) *A practical guide to SysML: the systems modeling language*, Morgan Kaufmann.

FRITZSON, P., ARONSSON, P., POP, A., LUNDVALL, H., NYSTROM, K., SALDAMLI, L., BROMAN, D. & SANDHOLM, A. (2006) OpenModelica-A free open-source environment for system modeling, simulation, and teaching. *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE.* IEEE.

FUH, J. Y. H. & LI, W. D. (2005) Advances in collaborative CAD: the-state-of-the art. *Computer-Aided Design,* 37**,** 571-581.

GABBERT, U. & WEHNER, P. (1998) The product data model as a pool for CAD-FEA data. *Engineering with Computers,* 14**,** 115-122.

GALLAHER, M. P., O'CONNOR, A. C. & PHELPS, T. (2002) Economic impact assessment of the international Standard for the Exchange of Product model data (STEP) in transportation equipment industries. *RTI International and National Institute of Standards and Technology, Gaithersburg, MD*.

GAO, S., ZHAO, W., LIN, H., YANG, F. & CHEN, X. (2010) Feature suppression based CAD mesh model simplification. *Computer-Aided Design,* 42**,** 1178-1188.

GARBADE, R. & DOLEZAL, W. (2007) DMU@Airbus: Evolution of the Digital Mock-up (DMU) at Airbus to the Centre of Aircraft Development. *The Future of Product Development.* Springer.

GERBINO, S. (2003) Tools for the interoperability among CAD systems. *Proc. XIII ADM-XV INGEGRAF Int. Conf. Tools and Methods Evolution in Engineering Design.*

GERO, J. S. (1990) Design prototypes: a knowledge representation schema for design. *AI magazine,* 11**,** 26.

GERO, J. S. & KANNENGIESSER, U. (2004) The situated function-behaviour-structure framework. *Design Studies,* 25**,** 373-391.

GERSHENSON, J. K., PRASAD, G. J. & ZHANG, Y. (2003) Product modularity: Definitions and benefits. *Journal of Engineering Design,* 14**,** 295-313.

GIELINGH, W. (2008) An assessment of the current state of product data technologies. *Computer-Aided Design,* 40**,** 750-759.

GIFFIN, M., KELLER, R., ECKERT, C., DE WECK, O., BOUNOVA, G. & CLARKSON, P. J. (2009) Change Propagation Analysis in Complex Technical Systems. *Journal of Mechanical Design,* 131**,** 081001-081001.

GOMES, S. & SAGOT, J. (2002) A CONCURRENT ENGINEERING EXPERIMENT BASED ON A CO-OPERATIVE AND OBJECT ORIENTED DESIGN METHODOLOGY. *Integrated Design and Manufacturing in Mechanical Engineering: Proceedings of the Third Idmme Conference Held in Montreal, Canada, May 2000.* Springer.

GRAF, H., BRUNETTI, G. & STORK, A. (2002) CAD2VR or how to efficiently integrate VR into the product development process. *CAD 2002: Corporate Engineering Research.* GI German Informatics Society.

GRAIGNIC, P., VOSGIEN, T., JANKOVIC, M., TULOUP, V., BERQUET, J. & TROUSSIER, N. (2013) Complex System Simulation: Proposition of a MBSE Framework for Design-Analysis Integration. *Procedia Computer Science,* 16**,** 59-68.

GRIFFIN, A. (1993) Metrics for measuring product development cycle time. *Journal of Product Innovation Management,* 10**,** 112-125.

GRIFFIN, A. (1997) Modeling and measuring product development cycle time across industries. *Journal of Engineering and Technology Management,* 14**,** 1-24.

GUJARATHI, G. P. & MA, Y.-S. (2010) Generative CAD and CAE integration using common data model. *Automation Science and Engineering (CASE), 2010 IEEE Conference on,* 21-24 Aug. 2010, Toronto, Canada.

GUJARATHI, G. P. & MA, Y. S. (2011) Parametric CAD/CAE integration using a common data model. *Journal of Manufacturing Systems,* 30**,** 118-132.

GUYOT, E., DUCELLIER, G., EYNARD, B., GIRARD, P. & GALLET, T. (2007) Product data and digital mock-up exchange based on PLM. *Proceedings of PLM07***,** 243-252.

HAMDI, M., AIFAOUI, N. & BENAMARA, A. (2007) Idealization of CAD Geometry Using Design and Analysis Integration Features Models. IN BOCQUET, J.-C. (Ed.) *ICED07: 16th International Conference of Engineering Design.* Paris, France, Design Society.

HAMILTON, B. A. (2010) Systems-2020 Study Final Report.

HAMRAZ, B., CALDWELL, N. H. M. & JOHN CLARKSON, P. (2012) A Multidomain Engineering Change Propagation Model to Support Uncertainty Reduction and Risk Management in Design. *Journal of Mechanical Design,* 134**,** 100905-100905.

HAMRI, O., LÉON, J. C., GIANNINI, F., FALCIDIENO, B., POULAT, A. & FINE, L. (2008) Interfacing product views through a mixed shape representation. Part 1: Data structures and operators. *International Journal on Interactive Design and Manufacturing (IJIDeM),* 2**,** 69-85.

HERZIG, S., QAMAR, A., REICHWEIN, A. & PAREDIS, C. J. (2011) A conceptual framework for consistency management in model-based systems engineering. *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2011.* ASME.

HERZOG, E. (2004) An approach to systems engineering tool data representation and exchange. *Department of Computer and Information Science.* Linköping, Sweden, Linköping universitet.

HOFFMAN, C. M. & JOAN-ARINYO, R. (1998) CAD and the product master model. *Computer-Aided Design,* 30**,** 905-918.

HOFFMANN, C. M. & JOAN-ARINYO, R. (2000) Distributed maintenance of multiple product views. *Computer-Aided Design,* 32**,** 421-431.

HOPPMANN, J. (2009) The Lean Innovation Roadmap - A Systematic Approach to Introducing Lean in Product Development Processes and Establishing a Learning Organization. Braunschweig, Germany, Technical University of Braunschweig & Massachusetts Institute of Technology.

HUNTEN, K. A. (1997) CAD/FEA integration with STEP AP209 technology and implementation. *IPT Lead - Analysis Tools and Integration, Virtual Product Development Initiative.* Lockheed Martin Corporation.

IACOB, R., MITROUCHEV, P. & LÉON, J.-C. (2008) Contact identification for assemblyâ€“disassembly simulation with a haptic device. *The Visual Computer,* 24**,** 973-979.

IEEE, I. S. A.-S. S. E. S. C. (2005) IEEE 1220: Standard for Application and Management of the Systems Engineering Process.

INCOSE (2008) Survey of Model-Based Systems Engineering (MBSE) Methodologies INCOSE.

IRLINGER, R. (1999) *Methoden und Werkzeuge zur nachvollziehbaren Dokumentation in der Produktentwicklung*, Shaker.

ISO-SCRA (2006) STEP APPLICATION HANDBOOK ISO 10303 VERSION 3

ISO (1994a) ISO 10303-1: Overview and Fundamental Principles. *Industrial Automation Systems and Integration - Product Data Representation and Exchange.* International Standard Organization, Geneva, Switzerland.

ISO (1994b) ISO 10303-11: Description methods - The EXPRESS language reference manual. *Industrial automation systems and integration - Product data representation and exchange.* International Standard Organization, Geneva, Switzerland.

ISO (1999) ISO 10303-209: Application protocol: Composite and metallic structural analysis and related design. *Industrial Automation Systems and Integration - Product Data Representation and Exchange. .* International Standard Organization, Geneva, Switzerland.

ISO (2000a) ISO 10303-104: Integrated application resource: Finite element analysis. *Industrial automation systems and integration -- Product data representation and exchange.* International Organization for Standardization, Geneva, Switzerland.

ISO (2000b) ISO 10303-203. AP214Ed2: Core data for automotive mechanical design processes. *Industrial automation systems and integration - Product data representation and exchange.* International Standard Organization, Geneva, Switzerland.

ISO (2003a) ISO 10303-42: Geometric and topological representation. *Industrial automation systems and integration -- Product data representation and exchange.* International Standard Organization, Geneva, Switzerland.

ISO (2003b) ISO 10303-109: Kinematic and geometric constraints for assembly models. *Industrial automation systems and integration - Product data representation and exchange -.* International Standard Organization, Geneva, Switzerland.

ISO (2004a) ISO10303-44: Product structure configuration. *Industrial automation systems and integration - Product data representation and exchange.* International Standard Organization, Geneva, Switzerland.

ISO (2004b) ISO 10303-239: Product life cycle support. *Industrial automation systems and integration — Product data representation and exchange* International Organization for Standardization, Geneva, Switzerland.

ISO (2005) ISO 10303-203. AP203Ed2: Configuration controlled 3D designs of mechanical Parts and assemblies. *Industrial automation systems and integration — Product data representation and exchange.* International Standard Organization, Geneva, Switzerland.

ISO (2008) ISO/IEC 15288: Systems and software engineering -- System life cycle processes. International Standard Organization, Geneva, Switzerland.

ISO (2011a) ISO10303-43: Representation structures. *Industrial automation systems and integration - - Product data representation and exchange --.* International Standard Organization, Geneva, Switzerland.

ISO (2011b) ISO 10303-52: Mesh-based topology. *Industrial automation systems and integration -- Product data representation and exchange.* International Organization for Standardization, Geneva, Switzerland.

ISO (2012) ISO 10303-233: Systems engineering. *Industrial automation systems and integration -- Product data representation and exchange --.* International Organization for Standardization, Geneva, Switzerland.

JACOBSON, I. (1992) *Object-oriented software engineering: a use case driven approach*, Pearson Education India.

JAYARAM, U., KIM, Y., JAYARAM, S., JANDHYALA, V. K. & MITSUI, T. (2004) Reorganizing CAD Assembly Models (as-Designed) for Manufacturing Simulations and Planning (as-Built). *Journal of Computing and Information Science in Engineering,* 4**,** 98-108.

JOHNSON, J. (2000) The latest developments in design data exchange: towards fully integrated aerospace design environments. *Proceedings of 22nd International Congress of Aeronautical Sciences.*

JOHNSON, J., NILSSON, B., LUISE, F., TÖRNE, A., LOEUILLET, J.-L., CANDY, L., INDERST, M. & HARRIS, D. (1999) The future systems engineering data exchange standard STEP AP-233: Sharing the results of the SEDRES Project. International Council on Systems Engineering.

JUGULUM, R. & SAMUEL, P. (2010) *Design for lean six sigma: A holistic approach to design and innovation*, Wiley. com.

KATO, J. (2005) Development of a process for continuous creation of lean value in product development organizations. Massachusetts Institute of Technology.

KEMMERER, S. J. (1999) *STEP: the grand experience*, US Department of Commerce, Technology Administration, National Institute of Standards and Technology.

KIBAMBA, Y. (2011) Spécification et développement d'un environnement collaboratif de gestion de cycle de vie des données de simulation numérique. *Thèse de doctorat, Université de Technologie de Compiègne.*

KIM, J. & WILEMON, D. (2003) Sources and assessment of complexity in NPD projects. *R&D Management,* 33**,** 15-30.

KIM, K.-Y., WANG, Y., MUOGBOH, O. S. & NNAJI, B. O. (2004) Design formalism for collaborative assembly design. *Computer-Aided Design,* 36**,** 849-871.

KINDLER, E. & WAGNER, R. (2007) Triple graph grammars: Concepts, extensions, implementations, and application scenarios. *University of Paderborn*.

KLAAS, O. & SHEPARD, M. S. (2001) Embedding Reliable Numerical Analysis Capabilities into an Enterprise-Wide Information System. *Engineering with Computers,* 17**,** 151-161.

KLEINER, S., ANDERL, R. & GRÄB, R. (2003) A collaborative design system for product data integration. *Journal of Engineering Design,* 14**,** 421-428.

KNOWLEDGE INSIDE (2013) arKItect Designer.

KONING, H. & VAN VLIET, H. (2006) A method for defining IEEE Std 1471 viewpoints. *Journal of Systems and Software,* 79**,** 120-131.

KRAUSE, F. L. & KAUFMANN, U. (2007) Meta-Modelling for Interoperability in Product Design. *CIRP Annals - Manufacturing Technology,* 56**,** 159-162.

KWANGHOON, K., MCMAHON, C. A. & LEE, K. H. (2003) Design of a feature-based multi-viewpoint design automation system. *International Journal of CAD/CAM,* 3.

LABROUSSE, M., PERRY, N. & BERNARD, A. (2004) Modèle FBS-PPR: des objets d'entreprise\a la gestion dynamique des connaissances industrielles. *Gestion dynamique des connaissances industrielles.* Lavoisier.

LALOR, P. (2007) Simulation Lifecycle Management - Opens a New Window on the Future of Product Design and Manufacturing. NAFEMS.

LEAL, D., PASHLEY, C. D., VAUGHAN, C. & CHILCOTT, S. (1999) The Engineering Analysis Core Model: A 'plain man's guide' Version 5.0. PDES, Inc. .

LÉON, J.-C. (1999) Multi-views and multi-representations design framework applied to a preliminary design phase. IN CHEDMAIL, P., ET AL. (Ed.) *Integrated Design and Manufacturing in Mechanical Engineering (IDMME) conference.* Kluwer.

LÉON, J.-C. & FINE, L. (2005) A new approach to the Preparation of models for F.E. analyses. *International Journal of Computer Applications in Technology,* Vol 3, No 2/3/4**,** 166 - 184.

LI, W. D., LU, W. F., FUH, J. Y. H. & WONG, Y. S. (2005) Collaborative computer-aided design: research and development status. *Computer-Aided Design,* 37**,** 931-940.

LI, W. D., ONG, S. K., FUH, J. Y. H., WONG, Y. S., LU, Y. Q. & NEE, A. Y. C. (2004) Feature-based design in a distributed and collaborative environment. *Computer-Aided Design,* 36**,** 775-797.

LI, Y., WAN, L. & XIONG, T. (2011) Product data model for PLM system. *The International Journal of Advanced Manufacturing Technology,* 55**,** 1149-1158.

LINDEMANN, U., MAURER, M. & BRAUN, T. (2009) *Structural Complexity Management: An Approach for the Field of Product Design,* Berlin Heidelberg, Springer.

MAKEM, J. E., ARMSTRONG, C. G. & ROBINSON, T. T. (2012) Automatic decomposition and efficient semi-structured meshing of complex solids. *Proceedings of the 20th International Meshing Roundtable.* Springer.

MARCA, D. A. & MCGOWAN, C. L. (1993) *IDEFO-SADT Business Process and Enterprise Modelling*, Eclectic Solutions Corporation.

MC CARTY, T., BREMER, M., DANIELS, L., GUPTA, P., HEISEY, J. & MILLS, K. (2005) *The Six Sigma black belt handbook*, McGraw-Hill New York.

MC MANUS, H. L., HAGGERTY, A. & MURMAN, E. (2007) *Lean engineering : a framework for doing the right thing right,* London, ROYAUME-UNI, Royal Aeronautical Society.

MC MANUS, H. & MILLARD, R. (2004) Value stream analysis and mapping for product development.

MC NUTT, R. T. (1999) Reducing DoD Product Development Time: The Role of the Schedule Development Process. DTIC Document.

MEINADIER, J.-P. (1998) *Ingénierie et intégration des systèmes*, Hermes.

MEYER, M. H. & UTTERBACK, J. M. (1995) Product development cycle time and commercial success. *Engineering Management, IEEE Transactions on,* 42**,** 297-304.

MOCKO, G., MALAK, R., PAREDIS, C. & PEAK, R. (2004) A knowledge repository for behavioral models in engineering design. *24th ASME Computers and Information in Engineering Conference*, Salt Lake City, UT, September 28-October.

MOCKO, G. M. & FENVES, S. J. (2003) *A Survey of Design-Analysis Integration Issues.* NIST Interagency/Internal Report (NISTIR) - 6996.

MODELICA (2010) Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification Modelica Association.

MORGAN, J. M. (2002) High performance product development: a systems approach to a lean product development process. University of Michigan.

MORGAN, J. M. & LIKER, J. K. (2006) *The Toyota product development system*, Productivity press New York.

MÜLLER, D. & HEIMANNSFELD, K. (2000) Requirements Engineering Knowledge Management based on STEP AP233. *ProSTEP Science Days Conference Paper.*

MURMAN, E., REBENTISCH, E. & WALTON, M. (2000) Challenges in the better, faster, cheaper era of aeronautical design, engineering and manufacturing.

MURRAY, J. (2012) Model Based Systems Engineering (MBSE) Media Study International Council on System Engineering (INCOSE).

NARAYAN, L. (2008) *Computer aided design and manufacturing*, PHI Learning Pvt. Ltd.

NASA, N. A. A. S. A. (2007) NASA Systems Engineering Handbook. NASA.

NGUYEN VAN, T. (2006) System Engineering for collaborative data management systems: application to design / simulation loops. Thèse de doctorat, Laboratoire Génie Industriel, Chatenay Malabry, Ecole Centrale Paris.

NGUYEN VAN, T., FÉRU, F., GUELLEC, P. & YANNOU, B. (2006a) Engineering Data Management for extended enterprise-Context of the European VIVACE Project. *Product Lifecycle Management PLM-SP2.*

NGUYEN VAN, T., MAILLE, B. & YANNOU, B. (2006b) Digital Mock-Up – Capabilities and implementation in the PLM field. *International Conference on Product Lifecycle Management.* Bangalore, India, Inderscience Enterprises Ltd.

NOLAN, D., TIERNEY, C., ROBINSON, T. & ARMSTRONG, C. (2011) Defining Simulation Intent. *NAFEMS European Conference: Simulation Process and Data Management (SDM).* Munich, Germany.

NOLAN, D. C., TIERNEY, C. M., ARMSTRONG, C. G., ROBINSON, T. T. & MAKEM, J. E. (2013) Automatic dimensional reduction and meshing of stiffened thin-wall structures. *Engineering with Computers***,** *30*(4), 689-701.

OCC (2013) Open CASCADE Technology.
ODASD (2011) Office of the Deputy Assistant Secretary of Defense for Systems Engineering.
OH, Y., HAN, S.-H. & SUH, H. (2001) Mapping product structures between CAD and PDM systems using UML. *Computer-Aided Design,* 33**,** 521-529.
OMG (2010a) OMG Systems Modeling Language (OMG SysML™) Version 1.2. Object Management Group.
OMG (2010b) OMG Unified Modeling Language TM (OMG UML), Superstructure Version 2.3. Object Management Group.
PAHL, G., BEITZ, W., SCHULZ, H.-J. & JARECKI, U. (2007) *Engineering design: a systematic approach*, Springer.
PARDESSUS, T. (2001) The multi-site extended enterprise concept in the aeronautical industry. *Air & Space Europe,* 3**,** 46-48.
PAREDIS, C., BERNARD, Y., BURKHART, R. M., DE KONING, H.-P., FRIEDENTHAL, S., FRITZSON, P., ROUQUETTE, N. F. & SCHAMAI, W. (2010) An overview of the SysML-Modelica transformation specification. *2010 INCOSE International Symposium.*
PAREDIS, C. J., DIAZ-CALDERON, A., SINHA, R. & KHOSLA, P. K. (2001) Composable models for simulation-based design. *Engineering with Computers,* 17**,** 112-128.
PASQUAL, M. & WECK, O. (2012) Multilayer network model for analysis and management of change propagation. *Research in Engineering Design,* 23**,** 305-328.
PDES (2006) STEP Success Stories.
PEAK, R. S., BURKHART, R., FRIEDENTHAL, S., WILSON, M. W., BAJAJ, M. & KIM, I. (2007a) Simulation-based design using SysML - Part 1: a parametrics primer. *INCOSE intl. symposium, San Diego.*
PEAK, R. S., BURKHART, R. M., FRIEDENTHAL, S., WILSON, M. W., BAJAJ, M. & KIM, I. (2007b) Simulation-based design using SysML - Part 2: Celebrating diversity by example. *INCOSE intl. symposium, San Diego.*
PEAK, R. S., FULTON, R. E., NISHIGAKI, I. & OKAMOTO, N. (1999) Integrating engineering design and analysis using a multi-representation approach. *Engineering with Computers,* 14**,** 93-114.
PENG, T.-K. & TRAPPEY, A. J. C. (1998) A step toward STEP-compatible engineering data management: the data models of product structure and engineering changes. *Robotics and Computer-Integrated Manufacturing,* 14**,** 89-109.
POP, A., AKHVLEDIANI, D. & FRITZSON, P. (2007) Towards unified system modeling with the ModelicaML UML profile. *Proceedings of the 1st International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLTâ€™07).* Citeseer.
PRATT, M. J. (2005) ISO 10303, the STEP standard for product data exchange, and its PLM capabilities. *International Journal of Product Lifecycle Management,* 1**,** 86-94.
PRITCHARD, D. & MACPHERSON, A. (2007) Strategic destruction of the Western commercial aircraft sector: implications of systems integration and international risk-sharing business models. *Aeronautical Journal,* 111**,** 327-334.
PROSTEP_IVIP (2002) Usage Guide for the STEP PDM Schema V1.2, Release 4.3.
PYTHONOCC (2013) 3D CAD/CAE/PLM development framework for the Python programming language.
QUADROS, W. R., VYAS, V., BREWER, M., OWEN, S. J. & SHIMADA, K. (2010) A computational framework for automating generation of sizing function in assembly meshing via disconnected skeletons. *Engineering with Computers,* 26**,** 231-247.
RIEL, A. (2005) From DMU to BMU: Towards Simulation-Guided Automotive Powertrain Development. Vienna, Austria, Vienna University of Technology.
ROBINSON, T., ARMSTRONG, C. & FAIREY, R. (2011) Automated mixed dimensional modelling from 2D and 3D CAD models. *Finite Elements in Analysis and Design,* 47**,** 151-165.
ROSENMAN, M. A. & GERO, J. S. (1996) Modelling multiple views of design objects in a collaborative CAD environment. *Computer-Aided Design,* 28**,** 193-205.
RUMBAUGH, J. R., BLAHA, M. R., LORENSEN, W., EDDY, F. & PREMERLANI, W. (1990) Object-oriented modeling and design.
SADOUL, D. (2000) La maquette numérique : Un exemple de développement industriel. *Mécanique & Industries,* 1**,** 373-382.

SANDBERG, M., KOKKOLARAS, M., AIDANPÄÄ, J.-O., ISAKSSON, O. & LARSSON, T. (2009) A master-model approach to whole jet engine analysis and design optimization. *8th World Congress on Structural and Multidisciplinary Optimization June.*

SCHAMAI, W., FRITZSON, P., PAREDIS, C. & POP, A. (2009) Towards unified system modeling and simulation with ModelicaML: modeling of executable behavior using graphical notations. *Proceedings 7th Modelica Conference, Como, Italy.*

SCHWANKL, L. (2002) Analyse und Dokumentation in den frühen Phasen der Produktentwicklung. München, Technische Universität München.

SELLGREN, U. (1999) Simulation-driven design: motives, means, and opportunities. PhD Thesis, KTH, The Royal Institute of Technology, Stockholm, Sweden.

SELLGREN, U. (2006a) Interface modeling - a modular approach to identify and assess unintended product behavior. *NAFEMS 2nd Nordic Seminar: Prediction and Modelling of Failure Using FEA* Roskilde, Denmark.

SELLGREN, U. (2006b) A model-based approach to situated design reasoning with a PLM perspective. 1st Nordic Conference on Product Lifecycle Management, 25-26 January 2006, Göteborg, Sweden.

SELLGREN, U. (2009) The journey towards PLM manged and interface driven design. 2nd Nordic Conference on Product Lifecycle Management - NordPLM´09. Göteborg, Sweden.

SELLGREN, U. & DROGOU, R. (1998) A systems and process approach to behavior modeling in mechanical engineering. *Proc. Integrated design and Manufacturing in Mechanical Engineering, IDMME´98.* Compiègne, France.

SHAH, A. A., SCHAEFER, D. & PAREDIS, C. J. (2009) Enabling multi-view modeling with sysml profiles and model transformations. *International Conference on Product Lifecycle Management.*

SHAH, J. J. (1991) Assessment of features technology. *Computer-Aided Design,* 23**,** 331-343.

SHAHWAN, A., FOUCAULT, G., LÉON, J.-C. & FINE, L. (2011) Towards Automated Identification of Functional Designations of Components Based on Geometric Analysis of a DMU. *12èmes Journées du Groupe de Travail en Modélisation Géométrique (GTMG 2011), 30-31 march 2011, Grenoble, France.*

SHAHWAN, A., LÉON, J.-C., FINE, L. & FOUCAULT, G. (2012) Reasoning about functional properties of components based on geometrical descriptions. *9th International Symposium on Tools and Methods of Competitive Engineering (TMCE 2012).*

SHAHWAN, A., LÉON, J.-C., FOUCAULT, G., TRLIN, M. & PALOMBI, O. (2013) Qualitative behavioral reasoning from components' interfaces to components' functions for DMU adaption to FE analyses. *Computer-Aided Design,* 45**,** 383-394.

SHEPHARD, M. S., BEALL, M. W., O'BARA, R. M. & WEBSTER, B. E. (2004) Toward simulation-based design. *Finite Elements in Analysis and Design,* 40**,** 1575-1598.

SINHA, R., PAREDIS, C., LIANG, V.-C. & KHOSLA, P. K. (2001a) Modeling and simulation methods for design of engineering systems. *Journal of Computing and Information Science in Engineering(Transactions of the ASME),* 123**,** 84-91.

SINHA, R., PAREDIS, C. J. & KHOSLA, P. K. (2000) Integration of mechanical CAD and behavioral modeling. *Behavioral Modeling and Simulation, 2000. Proceedings. 2000 IEEE/ACM International Workshop on.* IEEE.

SINHA, R., PAREDIS, C. J. & KHOSLA, P. K. (2001b) Interaction modeling in systems design. *Proceedings of DETC 2001, Computers in Engineering Conference.*

SINHA, R., PAREDIS, C. J. & KHOSLA, P. K. (2002) Behavioral model composition in simulation-based design. *Simulation Symposium, 2002. Proceedings. 35th Annual.* IEEE.

SODIUS (2013) Sodius - MDWorkbench.

SOSA, M. E., EPPINGER, S. D. & ROWLES, C. M. (2004) The misalignment of product architecture and organizational structure in complex product development. *Management science,* 50**,** 1674-1689.

SRINIVASAN, V. (2011) An integration framework for product lifecycle management. *Computer-Aided Design,* 43**,** 464-478.

STARK, J. (2011) Product Lifecycle Management. *Product Lifecycle Management: 21st century paradigm for product realisation.* London, Springer.

STEVENS, R. (1998) *Systems engineering: coping with complexity*, Pearson Education.

SUDARSAN, R., BAYSAL, M., ROY, U., FOUFOU, S., BOCK, C., FENVES, S., SUBRAHMANIAN, E., LYONS, K. & SRIRAM, R. (2005a) Information models for product representation: core and assembly models. *International Journal of Product Development,* 2**,** 207-235.

SUDARSAN, R., FENVES, S. J., SRIRAM, R. D. & WANG, F. (2005b) A product information modeling framework for product lifecycle management. *Computer-Aided Design,* 37**,** 1399-1411.

SUDARSAN, R., SUBRAHMANIAN, E., BOURAS, A., FENVES, S. J., FOUFOU, S. & SRIRAM, R. D. (2008) Information sharing and exchange in the context of product lifecycle management: Role of standards. *Computer-Aided Design,* 40**,** 789-800.

SUDARSAN, R., YOUNG-HYUN, H., SEBTI, F., SHAW, C. F., UTPAL, R., FUJUN, W., RAM D., S. & KEVIN W., L. (2006) A model for capturing product assembly information. *Journal of Computing and Information Science in Engineering,* 6**,** 11.

SUH, N. P. (2005a) Complexity in Engineering. *CIRP Annals - Manufacturing Technology,* 54**,** 46-63.

SUH, N. P. (2005b) *Complexity: theory and applications*, Oxford University Press, USA.

TABASTE, O. (2005) EDM FRAMEWORK DEFINITION. VIVACE Consortium.

TASSEY, G., BRUNNERMEIER, S. B. & MARTIN, S. A. (1999) Interoperability cost analysis of the US automotive supply chain. *Research Triangle Institute Final Report. RTI Project***,** 7007-03.

TICHKIEWITCH, S. (1996) Specifications on integrated design methodology using a multi-view product model. *The 1996 3 rd Biennial Joint Conference on Engineering Systems Design and Analysis, ESDA. Part 8(of 9).*

TICHKIEWITCH, S. & VÉRON, M. (1997) Methodology and product model for integrated design using a multiview system. *CIRP Annals-Manufacturing Technology,* 46**,** 81-84.

TOCHE, B., PELLERIN, R., FORTIN, C. & HUET, G. (2012) Set-Based Prototyping with Digital Mock-Up Technologies. *Product Lifecycle Management. Towards Knowledge-Rich Enterprises.* Springer Berlin Heidelberg.

TROUSSIER, N. (1999) Contribution à l'intégration du calcul mécanique dans la conception de produits techniques : proposition méthodologique pour l'utilisation et la réutilisation. Saint-Martin-d'Hères, FRANCE, Université de Grenoble 1.

ULRICH, K. T., EPPINGER, S. D. & GOYAL, A. (2011) *Product design and development*, Irwin/McGraw-Hill.

VOTINTSEVA, A., WITSCHEL, P., REGNAT, N. & STELZIG, P. (2012) Comparative Study of Model-Based and Multi-Domain System Engineering Approaches for Industrial Settings. *Modelling Foundations and Applications.* Springer Berlin Heidelberg.

VOIRIN, J.-L. (2010) METHOD & TOOLS TO SECURE AND SUPPORT COLLABORATIVE ARCHITECTING OF CONSTRAINED SYSTEMS *27th Congress of the International Council of the Aeronautical Sciences.* Nice, France, ICAS.

WANG, F., FENVES, S. J., SUDARSAN, R. & SRIRAM, R. D. (2003) Towards modeling the evolution of product families. *Proceedings of 2003 ASME DETC, September 2–6, 2003, Chicago, USA.*

WEINBERG, G. M. (1975) *An introduction to general systems thinking*, Wiley New York.

WENZEL, S. & BAUCH, T. (1996) Concurrent Engineering and More-Essentials for Successful Product Development from International Case Studies. Masters Thesis. Fachgebiet Raumfahrttechnik. Techn. Univ. Miinchen.

WILLAERT, S. S. A., DE GRAAF, R. & MINDERHOUD, S. (1998) Collaborative engineering: A case study of Concurrent Engineering in a wider context. *Journal of Engineering and Technology Management,* 15**,** 87-109.

WILLIAM XU, X. & LIU, T. (2003) A web-enabled PDM system in a collaborative design environment. *Robotics and Computer-Integrated Manufacturing,* 19**,** 315-328.

WOMACK, J. P. & JONES, D. T. (2003) *Lean thinking: banish waste and create wealth in your corporation, revised and updated*, Free Press.

WOMACK, J. P., JONES, D. T. & ROOS, D. (1990) The Machine that Changed the World: Based on the MIT 5-million-dollar 5-year Study on the Future of the Automobile. *New York: Rawson Associates*.

WOYAK, S. (2010) Simulation Driven Design: Creating an Environment for Managing Simulation Tools, Processes, and Data. Phoenix Integration.

YAN, X.-T. (2003) A multiple perspective product modeling and simulation approach to engineering design support. *Concurrent Engineering,* 11**,** 221-234.

YASSINE, A., WHITNEY, D., DALEIDEN, S. & LAVINE, J. (2003) Connectivity maps: Modeling and analysing relationships in product development processes. *Journal of Engineering Design,* 14**,** 377-394.

YOSHIOKA, M. & TOMIYAMA, T. (1997) Pluggable metamodel mechanism: A framework of an integrated design object modelling environment. *Computer Aided Conceptual Design97, Proceedings of the 1997 Lancaster International Workshop on Engineering Design {CACD}'97.*

ZHENG, X., SUN, G. & WANG, S. (2006) An Approach of Virtual Prototyping Modeling in Collaborative Product Design. *Computer Supported Cooperative Work in Design II.* Springer Berlin Heidelberg.

ZHOU, W. X., YIN, X. F., HSIUNG, C. H., FULTON, R. E., YEH, C.-P. & WYATT, K. (1997) CAD-Based Analysis tools for Electronic Packaging Design (A new modeling methodology for Virtual development Environment. *InterPACK'97, ASME.* Kohala Coast, Hawai.

APPENDIX I

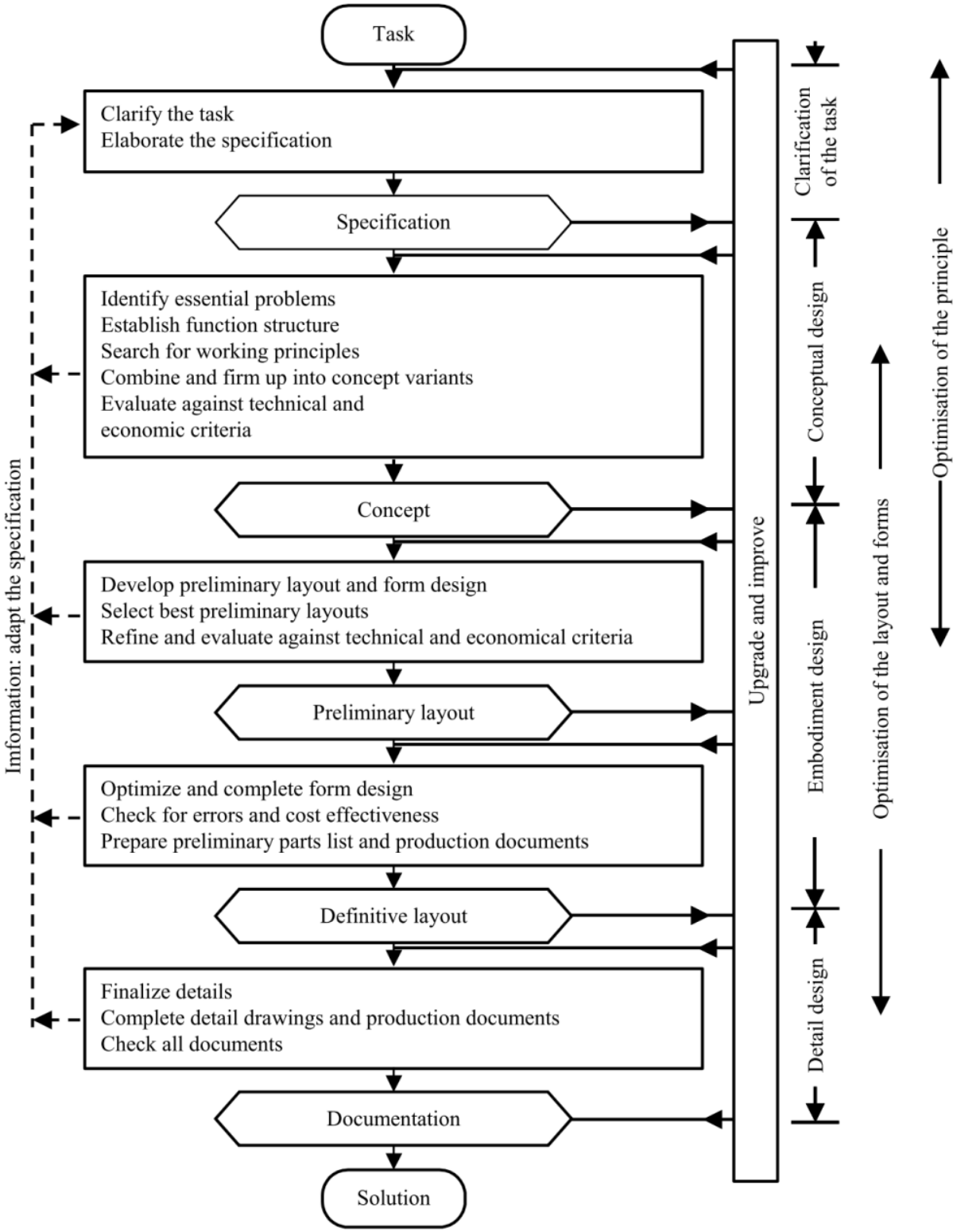


Figure 241: Generic Product Development Process according to [Pahl et al., 2007]

APPENDIX II

SE processes are defined by the norms IEEE 1220 and ANSI/EIA-632 and structure in four sequential sub-processes. As detailed in Table 16 and illustrated in Figure 242, the fundamental systems engineering activities are **Requirements Analysis, Functional Analysis and Physical Synthesis**—all balanced by techniques and tools collectively called **System Analysis and Control**. Systems engineering controls are used to track decisions and requirements, maintain technical baselines, manage interfaces, manage risks, track cost and schedule, track technical performance, verify requirements are met, and review/audit the progress. The System Analysis process includes all the trade study and assessment activities. It requires a global and multidisciplinary understanding of the system to identify, analyze and resolve conflicts in terms of requirements, choice of functional decomposition, allocation of performance requirements during the analysis functional, and in terms of selection of physical solutions. Each of these processes is followed by a process of verification / validation where one identifies and analyzes the differences and conflicts by comparing the results of phase with the entry requirements.

Phase	Activities
Requirements Analysis	Problem analysis and initial expression of needs and constraints,
	"External" functional analysis to define the functional and non-functional specification of the system that leads to the requirements book.
Functional Analysis	"Internal" functional analysis of potential solutions, from structural and behavioural point of view,
	Characterization of the solution (functional and non-functional specifications, constraints and relations on the parameters which characterize the system),
	Optimisation and evaluation of competing solutions.
Logical Synthesis	Defining the “logical” architecture of the system (kinematic schema, functional breakdown, logic diagrams, electrical, electronic schemas...)
	Setting a minimal behavioural model,
	Allowing a pre-sizing of the system to verify and validate the retained solution
Physical Synthesis	Defining the solution as an organic architecture supporting the functional architecture, and projecting the functions on existing components or on components to be developed,
	Specifying the interfaces and constituents needs remaining to design or/and to integrate,
	Leading to the global digital product model (DMU), multi-technologic, realistic and integrated,
	Checking the design and the behaviour of the components using such simulation software.

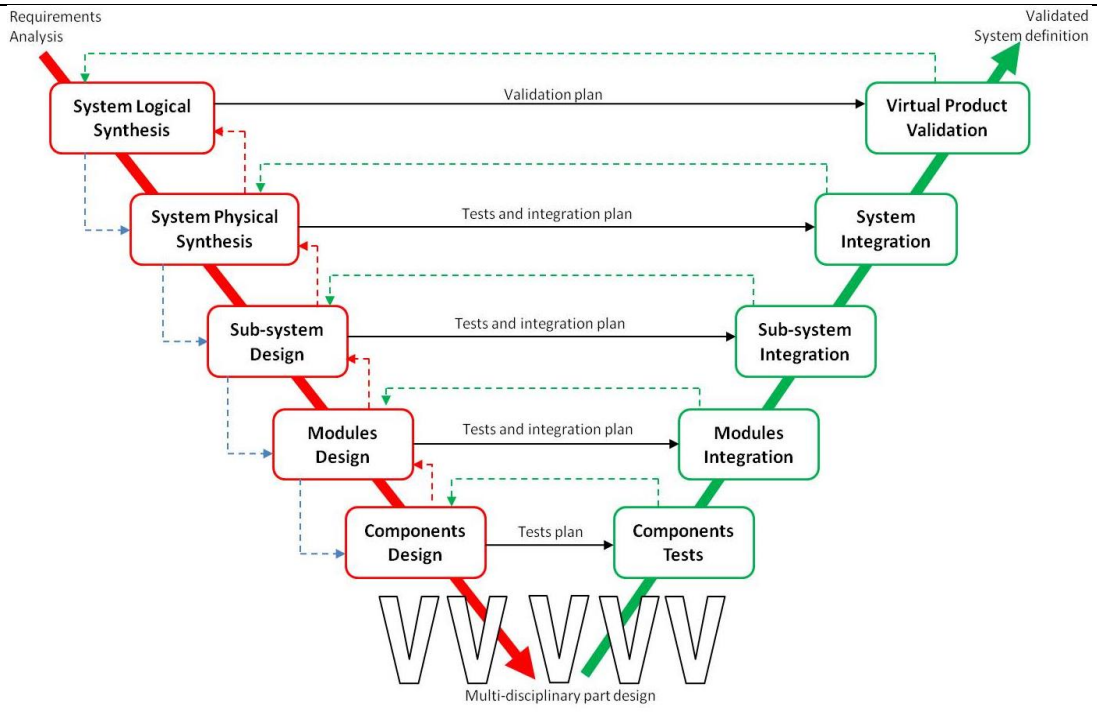
Table 16: System Engineering process phases and related activities (according to [IEEE, 2005])

SE approach is relevant when dealing with multiple breakdown system levels. It is hence inseparable from the notion of **System Integration**. These two complementary approaches and related processes are generally represented on the famous V cycle model. This V model establishes a logical arrangement of the development project activities. The V-cycle (Figure 242) is based on two dimensions:

- The decomposition level (vertically) or level of detail defined by the systematic finer and finer breakdowns of the system: the customer's needs level, the system level, the sub-systems level or modules level until the individual part level.
- The time (horizontally) between the tasks of the development project, from left to right.

These two dimensions hinge on a "V" with two branches (see Figure 242):

- A "design" descending branch where design activities are performed from a preliminary system definition (architecture, design space, interfaces, etc.) to a detailed components definition (real and accurate geometry and behaviour, physical interfaces, etc.) through the design of intermediate sub-systems or modules with an increasing granularity.
- An "integration" ascending branch composed of assembling, testing and validation activities including the testing/control of the designed or purchased components (parts). Then components' assemblies (minor modules), modules' assemblies (major modules or sub-system) and finally the whole system designs are successively tested and validated according to technical specifications (declined from customer's requirements).



Legend:

- **Feedback from integration to design** if the behavioural and/or dimensional analyses are not validated after assembly.
- **System or sub-systems architectures, technical specifications, interfaces specifications and change impact transmission.**
- **Refinement feedback** from design unit to higher design unit: if the preliminary design (geometry, design space, tolerances, behaviour, interfaces, etc.) of the system is not well defined regarding the detailed design of the sub-system.

Figure 242: System Engineering and Integration V cycle and the design iterations

APPENDIX III

In [Bauch, 2004], the authors introduce a model that draws a good parallel between the design processes activities and the information value creation process. Indeed, through these different transitions, information becomes more valuable, since it is more and more usable through this process [Bauch, 2004].

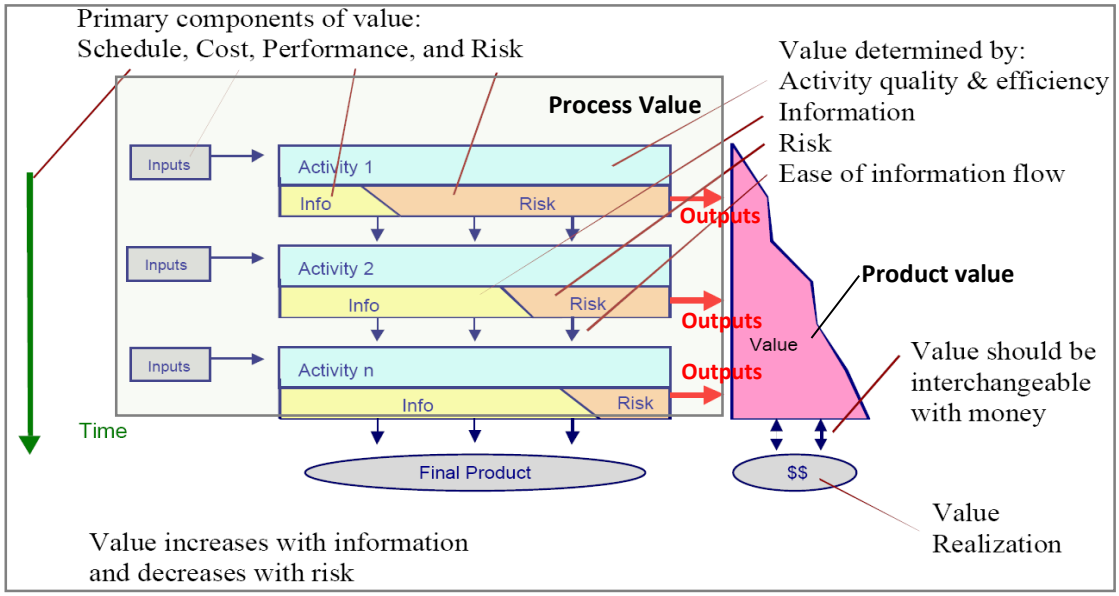


Figure 243: Conceptual framework for Value creation in PDP (adapted from [Chase, 2001])

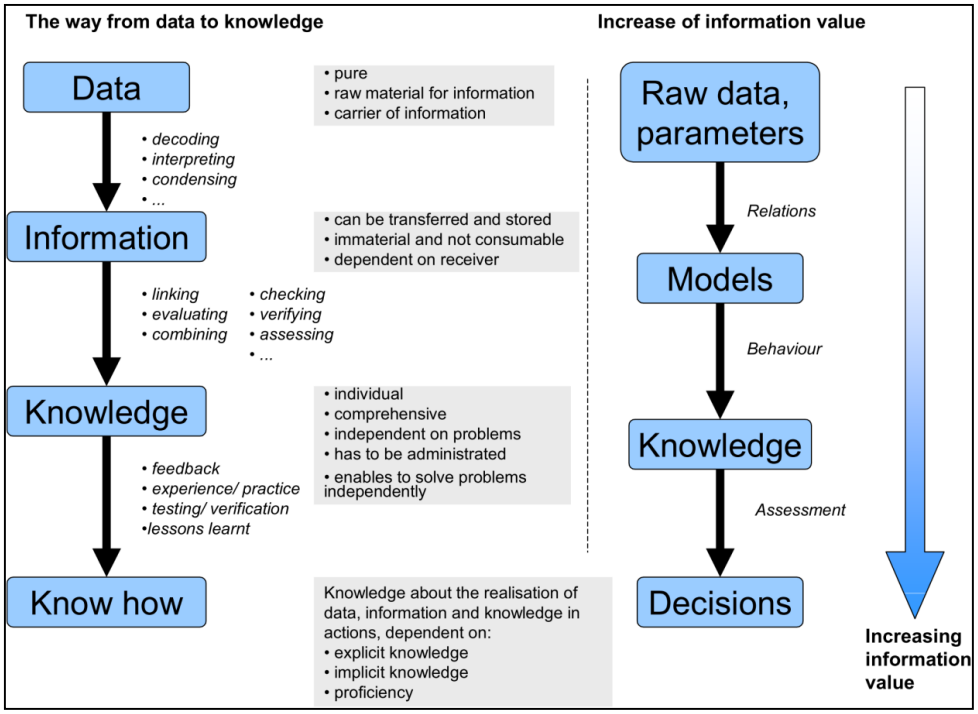


Figure 244: Data, information, knowledge and their value added (copied from [Bauch, 2004] but according to [Schwankl, 2002] and [Irlinger, 1999])

The last stage (knowledge and know-how) is supposed to enable designers to take good and rational decisions about the design choices. Information is an immaterial product. It can be precious for

the user, if it is taken into account in carrying out his tasks. Moreover, it is necessary to emphasize that the information value depends upon the context and the moment the information is used.

Waste definition and discussion in PDP

Bauch [Bauch, 2004], and afterwards Kato [Kato, 2005], map all the ideas from the previous studies in order to identify all potential type of wastes. Bauch elaborates a cause and effect diagram in order to distinguish waste types from waste drivers. We consider the categorization done by Bauch as a necessary basis to track down the waste in PDP. The author defines 6 categories of waste: Resources, Time, Information / Knowledge, Opportunity/ Potential, Money/Investments and Motivation. Since these 6 categories are strongly linked to the project targets, he adds to this list “project flexibility” and “quality to market”. He also defines 10 categories of waste drivers that encompass 37 sub-categories as shown in Figure 245 below.

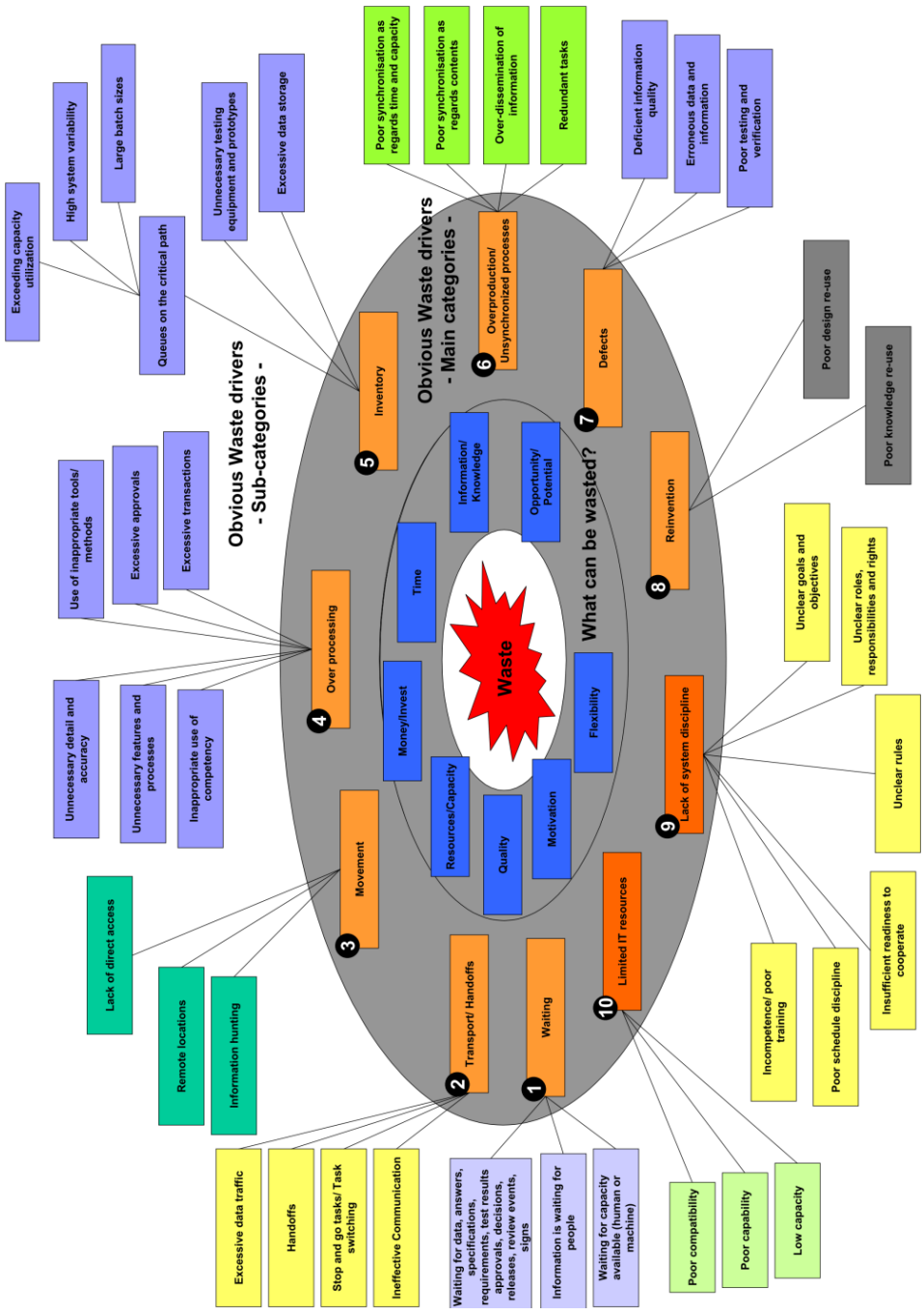


Figure 245: Overview of waste drivers in PDP [Bauch, 2004]

APPENDIX IV

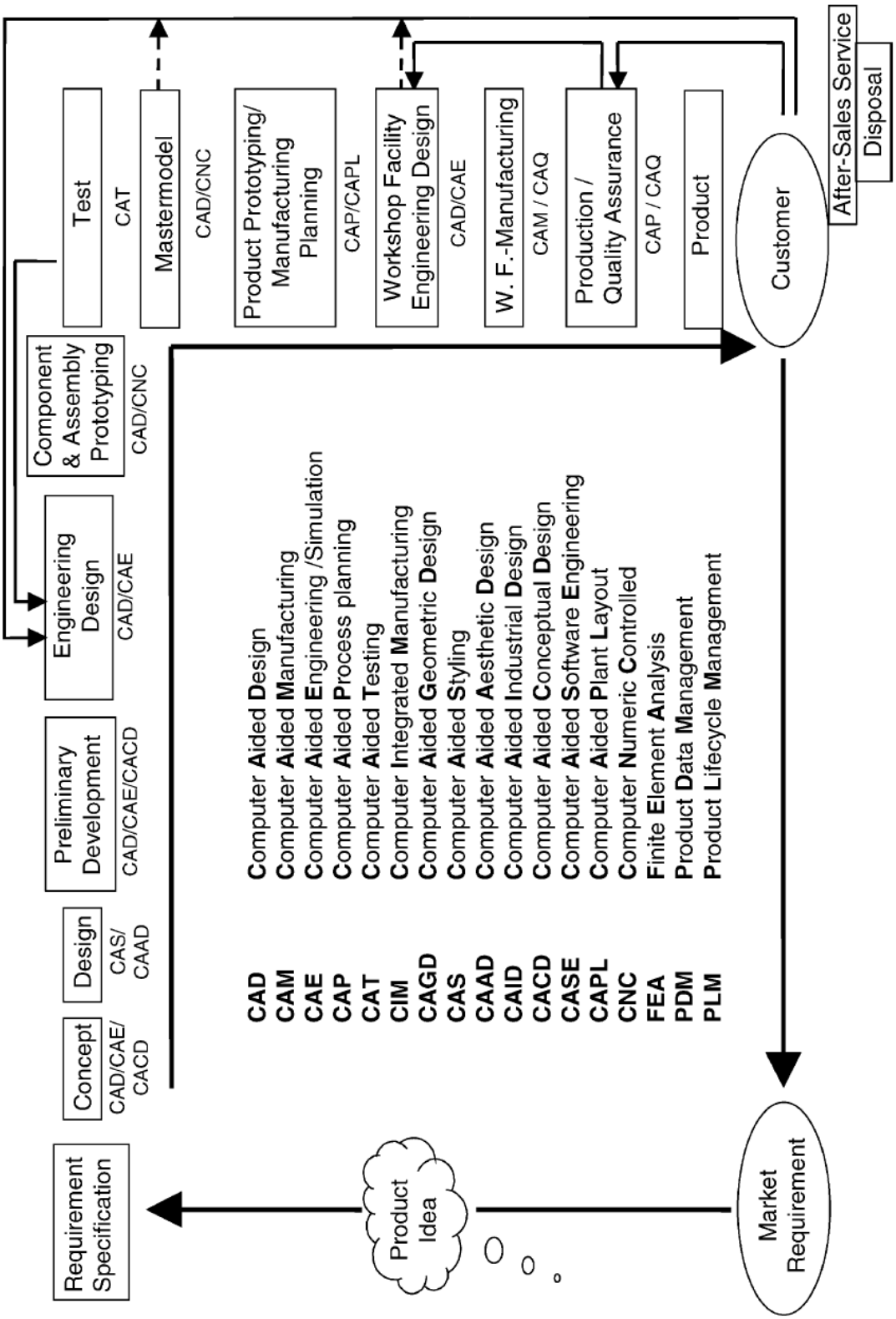


Figure 246: General process chain of product development with the corresponding used CAX technologies [Werner Dankwort et al., 2004]

APPENDIX V

Table 1
Standardized product data and product meta-data in ISO STEP AP 214.

	Class	Description	Remarks
Data (*Engineering objects*)	CC 1	Component design with 3D shape representation.	Covers 3D geometry of single parts, including wire-frame, surface, and solid models.
	CC 2	Assembly design with 3D shape representation.	Covers 3D geometry of assemblies of parts, including the assembly and model structure.
Meta-data (*Business objects*)	CC 6	Product data management (PDM) without shape representation.	Covers product data management systems that manage geometric models as files. It also covers administrative data of parts, assemblies, documents, and models.
	CC 8	Configuration controlled design without shape representation.	Covers CC 6, with additional requirements for product configuration control.

Table 2
Units of functionality covered by the STEP PDM Schema [24].

Units of Functionality	Content
Part identification	Part as Product
	Product master identification
	Context information
	Type classification
Specific part type classification	Classification of parts and managed documents
Part properties	General part properties
	External part shape
	External geometric model structure
	Relative orientation and location of related geometric models
Part structure and relationships	Explicit assembly Bill of Material
	Multi-level assembly digital mock-up
	Different views on assembly structure
	Relating part shape properties to product structure
	Other relationships between parts
Document identification	Document as Product
	Document master identification
	Context information
	Type classification
Specific document type classification	Product related product category and product category relationship
External files	External file identification
Relationship between documents and constituent files	Product definition with associated documents
Document and file properties	Product definition or document representation
	Document content property
	Document creation property
	Document size property
	Document source property
	Additional document properties
	Document type classification
Document and file association with product data	Document reference
	External file reference
	Constrained document or file reference
Document and file relationships	Sequence relationships between document versions
	Relationships between document representations
	Relationships between external files
Alias identification	Associating with additional identifier
Authorization	Organization and persons
	Approval
	Dates, times, and event references
	Security classification
	Certification
Configuration and effectivity information	Configuration identification
	Configuration composition management
	General validity period
Engineering change and work management	Request for work
	Work order and work definition
	Project identification
	Contract identification
Measure and units	Measures with unit specification
	Unit definition

Figure 247: Units of functionalities covered by the STEP PDM schema [Srinivasan, 2011]

APPENDIX VI

See: http://www.plcs.org/plcslib/plcslib/data/PLCS/concept_model/model_base.html

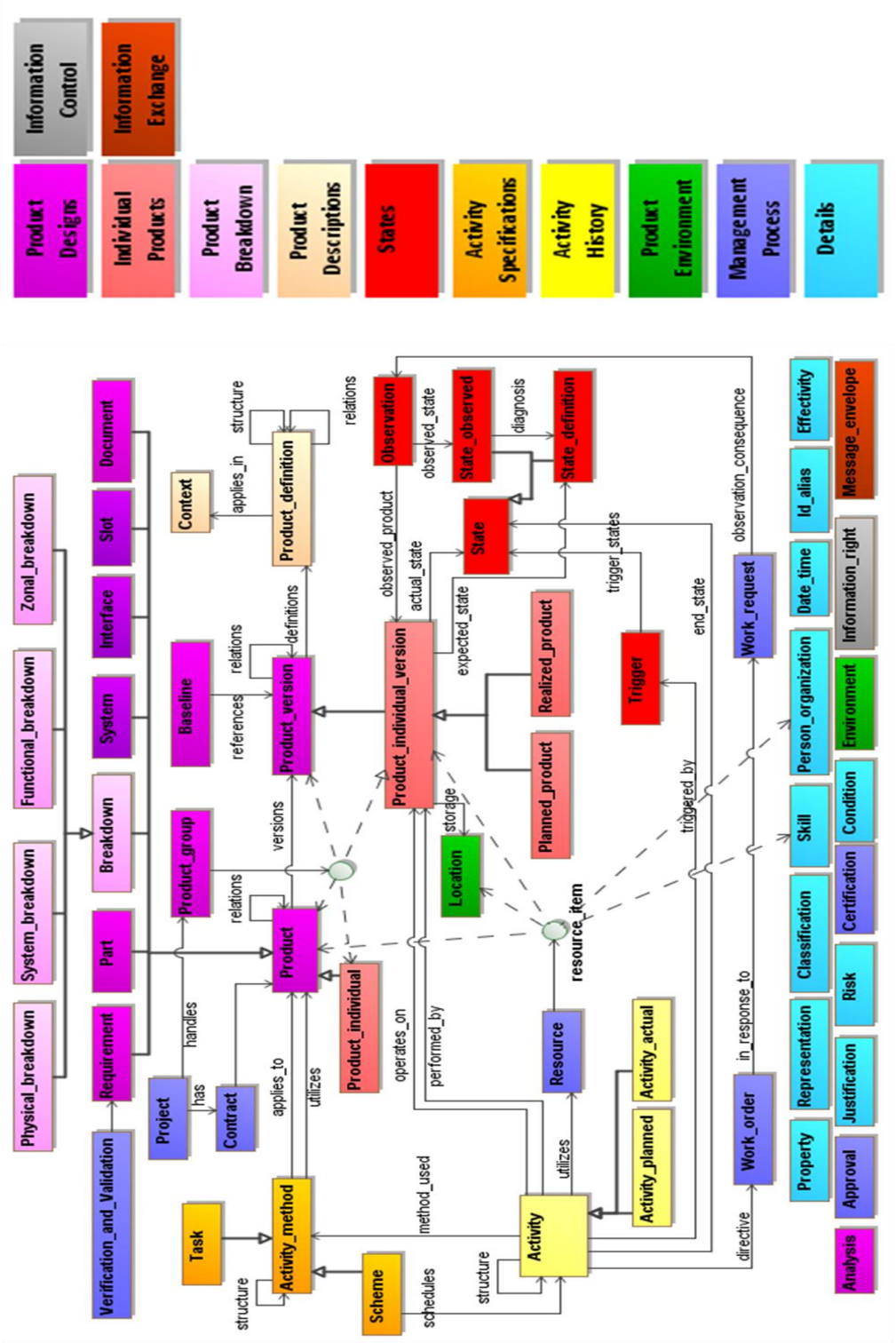


Figure 248: PLCS Conceptual Data Model [ISO, 2004b]

APPENDIX VII

Kinematic and geometric constraints for assembly models according to [ISO, 2003b]

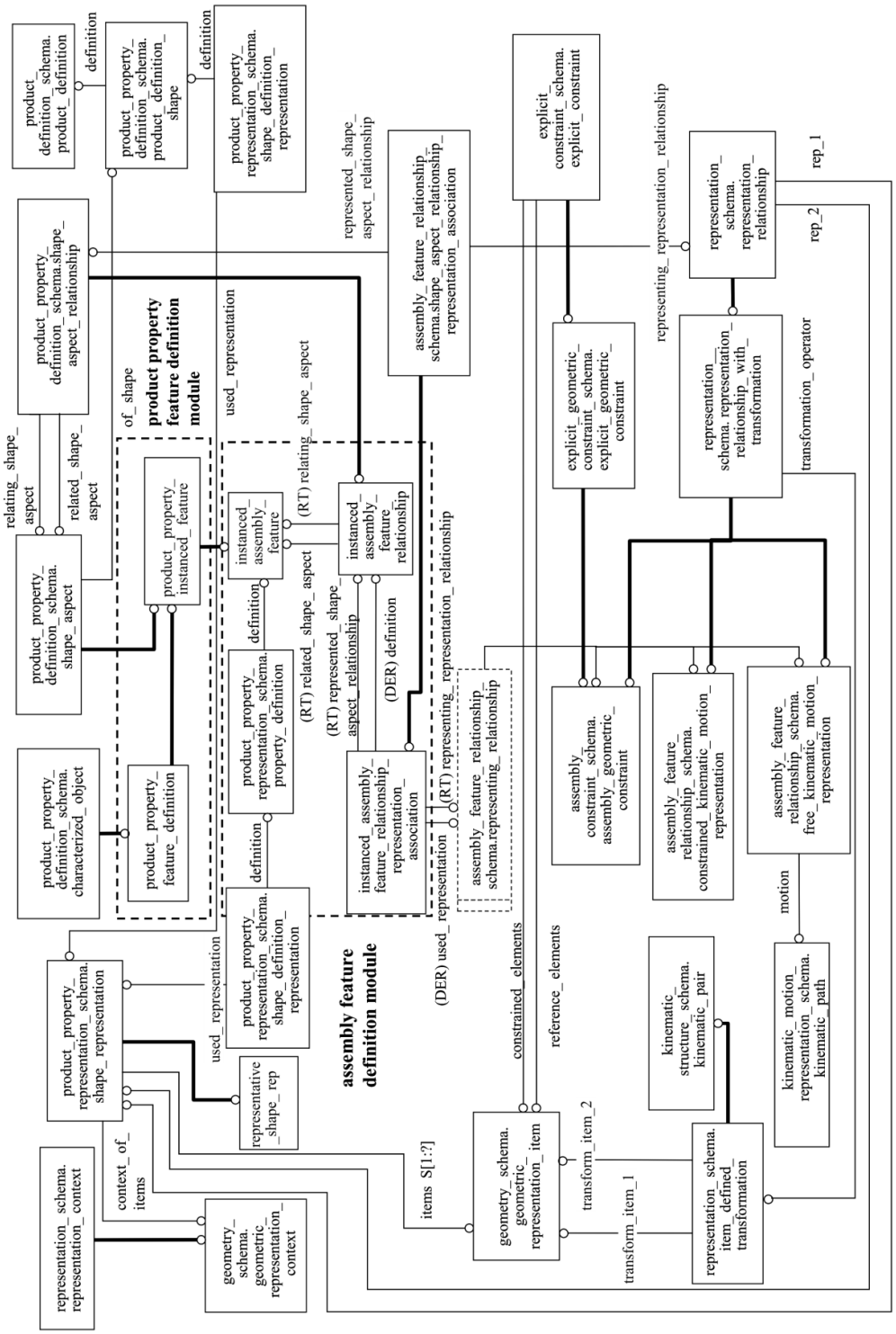


Figure 249: Express-G diagram of the Assembly_constraint_schema and Assembly_feature_relationship_schema [ISO, 2003b]

APPENDIX VIII

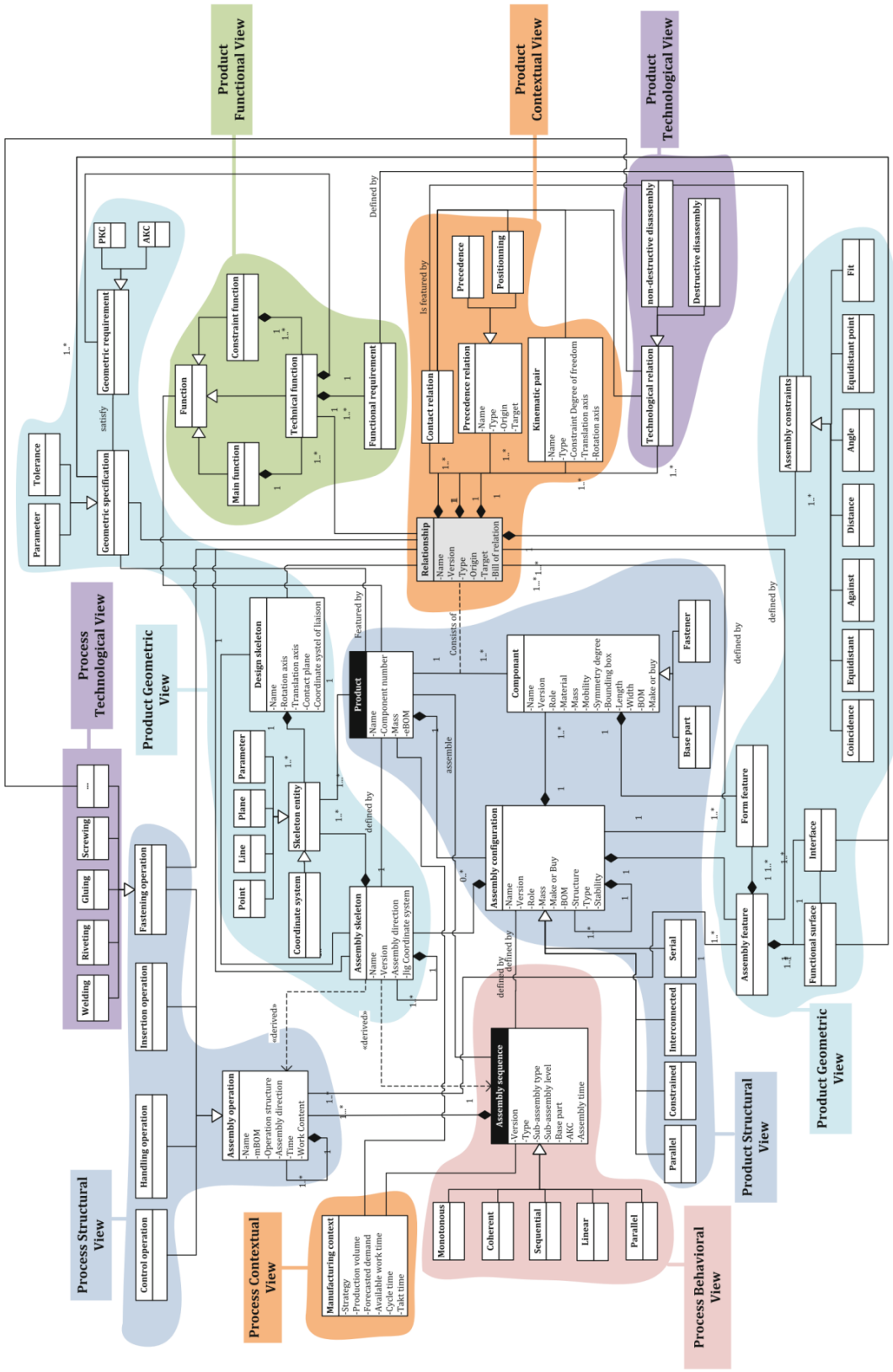


Figure 250: UML class diagram of the MUVOA model extracted from [Demoly et al., 2010b]

APPENDIX IX

The following table describes the MPC types which are supported for Marc (MSC Software).

MPC Type	Analysis Type	Description
Explicit	Structural Thermal Coupled	Creates a SERVO LINK explicit MPC between a dependent degree-of-freedom and one or more independent degrees-of-freedom. The dependent term consists of a node ID and a degree-of-freedom, while an independent term consists of a coefficient, a node ID, and a degree-of-freedom. An unlimited number of independent terms can be specified, while only one dependent term can be specified.
Rigid (Fixed)	Structural Coupled	Creates TYING Type 100 MPCs which constrains all degrees-of-freedom at one or more dependent nodes to the corresponding degrees-of-freedom at one independent node. An unlimited number of dependent terms can be specified, while only one independent term can be specified. Each term consists of a single node.
Linear Surf-Surf	Structural Coupled	Creates a TYING Type 31 MPC which constrains a dependent node on one linear 2D element to two independent nodes on another linear 2D element to model a continuum. One dependent term is specified, while two independent terms are specified. Each term consists of a single node.
Linear Surf-Surf	Thermal	Creates a TYING Type 87 MPC which constrains one dependent node to one independent node, which ties temperatures between shell elements. One dependent and one independent term are specified. A second independent term must be supplied but is ignored (it can be the same node). Each term consists of a single node.
Linear Surf-Vol	Thermal	Creates a TYING Type 85 MPC which constrains a dependent node on one linear 2D element to two independent nodes on another linear 2D element to tie temperatures. One dependent term is specified, while two independent terms are specified. Each term consists of a single node.
Linear Vol-Vol	Structural Thermal Coupled	Creates a TYING Type 33 MPC which constrains a dependent node on one linear 3D solid element to four independent nodes on another linear 3D solid element to model a continuum. One dependent term is specified, while four (three for degenerate face) independent terms must be specified. Each term consists of a single node.
Quad Surf-Surf (quadratic)	Structural Coupled	Creates a TYING Type 32 MPC which constrains a dependent node on one quadratic 2D element to three independent nodes on another quadratic 2D element to model a continuum. One dependent term is specified, while three independent terms are specified. Each term consists of a single node.
Quad Surf-Surf	Thermal	Identical to Linear Surf-Surf for Thermal analysis except a third independent term must be supplied but is also ignored.
Quad. Surf-Vol	Thermal	Creates a TYING Type 86 MPC which constrains a dependent node on one quadratic 2D element to three independent nodes on another quadratic 2D element to tie temperatures. One dependent term is specified, while three independent terms are specified. Each term consists of a single node.
Quad Vol-Vol	Structural Thermal Coupled	Creates a TYING Type 34 MPC which constrains a dependent node on one quadratic 3D solid to eight independent nodes on another quadratic 3D solid element to model a continuum. One dependent term is specified, while eight (six for degenerate face) independent terms are specified. Each term consists of a single node.
Tie DOFs	Structural Thermal Coupled	Creates a TYING Types 1-6 or 102-506 MPC which constrains two nodes at a selected degree-of-freedom or at a range of degrees-of-freedom. One dependent term is specified which consists of a single node. One independent term is specified which consists of a single node and either one or two selected degrees-of-freedom. The Marc type number will be determined by the selected degrees-of-freedom. If one degree-of-freedom is specified, a Type 1-6 MPC is created. If two degrees-of-freedom are selected, a Type 102-506 MPC is created.
Axi Shell-Solid	Structural Coupled	Creates a TYING Type 26 MPC which connects an axisymmetric shell element to a solid element. One dependent term is specified which consists of a single node. One independent term is specified which also consists of a single node.
Tri Plate-Plate	Structural Coupled	Creates a TYING Type 49 MPC which connects triangular flat plate elements. One dependent term is specified which consists of a single node. One independent term is specified which also consists of a single node.
Quad Plate-Plate	Structural Coupled	Creates a TYING Type 50 MPC which connects rectangular flat plate elements. One dependent term is specified which consists of a single node. One independent term is specified which also consists of a single node.

Pinned Joint	Structural Coupled	Creates a TYING Type 52 MPC which creates a pinned joint between beam elements. One dependent term is specified which consists of a single node. One independent term is specified which also consists of a single node.
Full Moment Joint	Structural Coupled	Creates a TYING Type 53 MPC which is a full moment joint between beam elements. One dependent term is specified which consists of a single node. One independent term is specified which also consists of a single node.
Rigid Link	Structural Coupled	Creates a TYING Type 80 MPC which creates a pinned rigid link between two nodes. One dependent term is specified, while two independent terms are specified. The dependent term and the first independent term are the nodes at the ends of the link, while the second independent term is an unattached node that provides the rotational information about the link.
Cyclic Symmetry	Structural Coupled	Creates a TYING Type 100 MPC which ties all degrees-of-freedom between matched nodes on opposite sides of the cyclic sector. Unlimited nodes may be entered in the dependent and independent regions; however, the same number of unique nodes must be specified in both regions.
Sliding Surface	Structural Coupled	Creates a SERVO LINK explicit MPC which ties the normal to the surface degrees-of-freedom between matched nodes on opposite sides of the interface. Unlimited nodes may be entered in the dependent and independent regions; however, the same number of unique nodes must be specified in both regions.
RBE2	Structural	Creates an MD Nastran style RBE2 element, which defines a rigid body between an arbitrary number of nodes. Although the user can only specify one dependent term, an arbitrary number of nodes can be associated to this term. The user is also prompted to associate a list of degrees of freedom to this term. A single independent term can be specified, which consists of a single node. There is no constant term for this MPC type. The RBE parameter is also written.
RBE3	Structural	Creates an MD Nastran style RBE3 element, which defines the motion of a reference node as the weighted average of the motions of a set of nodes. A finite number of dependent terms can be specified, each term consisting of a single node and a list of degrees of freedom. The first dependent (tied) term is used to define the reference node. Any (optional) dependent terms define additional nodes/degrees of freedom (dofs) that are added to the m-set. These additional dependent (tied) nodes/dofs MUST be a subset of the independent (retained) nodes/dofs as defined next. An arbitrary number of independent (retained) terms must also be specified. Each independent term consists of a constant coefficient (weighting factor), a node, and a list of degrees of freedom. All nodes with the same weighting factor and dof list should be grouped together. There is no constant term for this MPC type and at the present time, the Thermal Expansion coefficient is ignored. The RBE parameter is also written.
Overclosure	Structural Thermal Coupled	Creates a TYING Type 69 MPC which is used for creating gaps or overlaps between two parts of a model either by prescribing the total force on the nodes on either side of the gap/overlap or by prescribing the size of the gap/overlap. This is typically used for pretensioning of bolts or rivets. Dependent terms contain one node each and independent terms contain two nodes each. Each dependent (tied) term consists of a node on one side of the gap/overlap. The first node of the independent (retained) term consist of the corresponding node on the other side of the gap/overlap. The second node of the independent term is a control node to which LBCs may be applied. Each independent term must have the same control node otherwise an error is issued. There must be the same number of independent vs. dependent terms also, otherwise an error is issued. The control node should not be associated to any elements. In non-mechanical passes, this MPC reduces to a Type 100 between the dependent and first independent term internally to MSC.Marc.

Table 17: Finite Elements Multi-Point Constraints in the MSC Patran and Marc pre-processing tools[11]

[11] http://www.mscsoftware.com/training_videos/patran/Reverb_help/index.html#page/Marc/marc02_model.3.4.html#ww74264

APPENDIX X

```
<?xml version = "1.0" encoding = "UTF-8"?>
<XMI xmi.version = "1.1" xmlns:UML="href://org.omg/UML/1.3" timestamp = "Thu Apr 04 18:15:53 2013">
<XMI.header>
 <XMI.documentation>
  <XMI.owner></XMI.owner>
  <XMI.contact></XMI.contact>
  <XMI.exporter>StarUML.XMI-Addin</XMI.exporter>
  <XMI.exporterVersion>1.0</XMI.exporterVersion>
  <XMI.notice></XMI.notice>
 </XMI.documentation>
 <XMI.metamodel xmi.name = "UML" xmi.version = "1.3"/>
</XMI.header>
<XMI.content>
<UML:Model xmi.id="UMLProject.1">
 <UML:Namespace.ownedElement>
  <UML:Model xmi.id="UMLModel.2" name="data_model" visibility="public" isSpecification="false" namespace="UMLProject.1" isRoot="true" isLeaf="false" isAbstract="false">
   <UML:Namespace.ownedElement>
    <UML:Package xmi.id="UMLPackage.3" name="system_definition" visibility="public" isSpecification="false" namespace="UMLModel.2" isRoot="false" isLeaf="false" isAbstract="false">
     <UML:Namespace.ownedElement>
      <UML:Class xmi.id="UMLClass.4" name="Design_Artifact" visibility="public" isSpecification="false" namespace="UMLPackage.3" supplierDependency="UMLRealization.60 UMLRealization.254 UMLRealization.339 UMLRealization.466" isRoot="false" isLeaf="false" isAbstract="true" participant="UMLAssociationEnd.653 UMLAssociationEnd.292" isActive="false">
       <UML:Classifier.feature>
        <UML:Attribute xmi.id="UMLAttribute.5" name="name" visibility="public" isSpecification="false" ownerScope="instance" changeability="changeable" targetScope="instance" type="X.681" owner="UMLClass.4"/>
        <UML:Attribute xmi.id="UMLAttribute.6" name="type" visibility="public" isSpecification="false" ownerScope="instance" changeability="changeable" targetScope="instance" type="X.681" owner="UMLClass.4"/>
        <UML:Attribute xmi.id="UMLAttribute.7" name="instance_required" visibility="public" isSpecification="false" ownerScope="instance" changeability="changeable" targetScope="instance" type="X.684" owner="UMLClass.4">
         <UML:Attribute.initialValue>
          <UML:Expression xmi.id="X.685" body="true"/>
         </UML:Attribute.initialValue>
        </UML:Attribute>
        <UML:Operation xmi.id="UMLOperation.8" name="__init__" visibility="public" isSpecification="false" ownerScope="instance" isQuery="false" concurrency="sequential" isRoot="false" isLeaf="false" isAbstract="false" specification="" owner="UMLClass.4">
         <UML:BehavioralFeature.parameter>
          <UML:Parameter xmi.id="UMLParameter.9" name="name" visibility="public" isSpecification="false" kind="in" behavioralFeature="UMLOperation.8" type=""/>
          <UML:Parameter xmi.id="UMLParameter.10" name="type" visibility="public" isSpecification="false" kind="in" behavioralFeature="UMLOperation.8" type=""/>
          <UML:Parameter xmi.id="UMLParameter.11" name="instance_required" visibility="public" isSpecification="false" kind="in" behavioralFeature="UMLOperation.8" type=""/>
         </UML:BehavioralFeature.parameter>
        </UML:Operation>
        <UML:Operation xmi.id="UMLOperation.12" name="__repr__" visibility="public" isSpecification="false" ownerScope="instance" isQuery="false" concurrency="sequential" isRoot="false" isLeaf="false" isAbstract="false" specification="" owner="UMLClass.4"/>
        <UML:Operation xmi.id="UMLOperation.13" name="defs" visibility="public" isSpecification="false" ownerScope="instance" isQuery="false" concurrency="sequential" isRoot="false" isLeaf="false" isAbstract="false" specification="" owner="UMLClass.4"/>
        <UML:Operation xmi.id="UMLOperation.14" name="new_def" visibility="public" isSpecification="false" ownerScope="instance" isQuery="false" concurrency="sequential" isRoot="false" isLeaf="false" isAbstract="false" specification="" owner="UMLClass.4"/>
       </UML:Classifier.feature>
      </UML:Class>
...
```

Table 18: Extract of the XMI file of the logical DASIF prototype data model

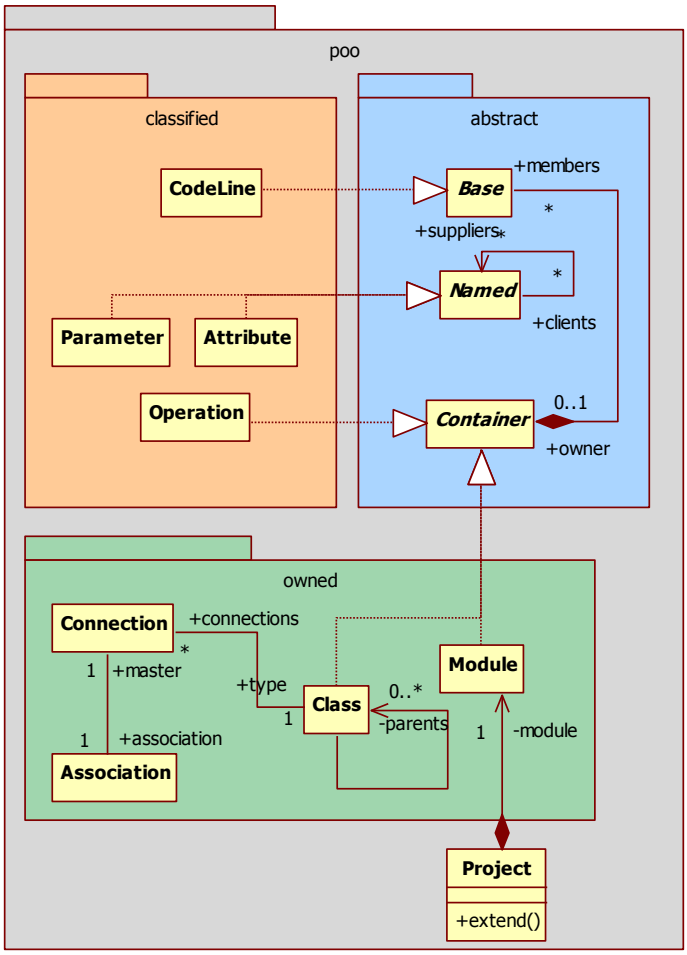


Figure 251: Simplified GENEPY neutral model

```
# -*- coding: utf-8 -*-
"""Empty documentation."""

# Import Elixir definitions
from elixir import (
   Boolean,
   Date,
   Entity,
   Field,
   Float,
   Integer,
   ManyToMany,
   ManyToOne,
   OneToMany,
   OneToOne,
   String,
   using_options_defaults )

# Setup Elixir
from datamodel import db
__metadata__ = db.metadata
__session__ = db.session
using_options_defaults(inheritance='multi')

import math
import numpy as np

import win32com.client
import os.path
import psutil
import datamodel
```

```
class DesignArtifact(Entity):
    """
    Description
    ===========
    This is the base abstract class for all the diffrent kind of
    elements/objects particpating in the definition of a system/product. It is
    a collector of data common to all versions of a system constituent.

    Therefore the instances of this class are the constituents / components of
    a system/product; wether they are items (parts or assemblies potentially
    intended to be produced), connectors (components that play the role of
    interfaces), structural assembly (enabling to structure the product) or
    even the fluid domains (that need to be considered in the definition of the
    product; especially for the definition of a turbo-machines).


    Fields
    ------
    name
        Empty documentation.

    type
        Empty documentation.

    instance_required
        Empty documentation.


    Relationships
    -------------
    defs
        List of definitions for the artefact.

    change
        The change that describes the modification of a design artefact
        definition.

    """
    name = Field(String(64))
    instance_required = Field(Boolean)
    defs = OneToMany('DesignDefinition', inverse='artifact')
    change = ManyToOne('DesignChange')


    def __init__(self, name):
        """
        Description
        ===========
        Construct an instance of Design Artefact.

        Parameters
        ----------
        name
            Empty documentation.

        type
            Empty documentation.

        instance_required
            Empty documentation.

        """
        # TODO

        self.name=name


    def __repr__(self):
        """
        Description
        ===========
        Return instance's representation: what the method print(artefact) returns.
        """
```

```
        # TODO
        return "<Artefact  Name= '{}'>".format(self.name)


    def get_defs(self):
        """
        Description
        ===========
        Return artefact's instance design definitions (definition versions).
        """
        # TODO
        raise NotImplementedError

    def new_def(self):
        """
        Description
        ===========
        Return a new instance of the artefact design_definition (new version).
        """
        # TODO
        raise NotImplementedError


class Item(DesignArtifact):
    """
    Description
    ===========
    The items are the tangible constituents of a system. They represents all
    artefacts that make system exist in a real world. An item intends to be the
    result of a manufacturing process.

    An item is a product that has its own characteristics. Example: A pencil is
    a product and there must be diffrent items defined for the same product
    (e.g. the red and the blue pencil).

    It is a specialization of the entity "CoreElement" since all  constituents
    involved in a product structure are not necessarily items intended to be
    produced; that is why we need another specialization of the CoreElement
    class which is the structural assembly class.


    Fields
    ------
    ref
        Empty documentation.

    index
        Empty documentation.


    Relationships
    -------------
    change_a
        Empty documentation.

    change_b
        Empty documentation.

    """
    ref = Field(String(64))
    index = Field(String(64))
    change_a = ManyToOne('AlternateItemRelationship')
    change_b = ManyToOne('AlternateItemRelationship')



    def __init__(self, name, ref, index):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
```

```
        ref
            Empty documentation.

        index
            Empty documentation.

        """
        # TODO
        DesignArtifact.__init__(self, name)
        self.ref=ref
        self.index=index


class ItemPart(Item):
    """
    Description
    ===========
    A Part is a single item (piece) generally considered as undismantled.
    However we can distinguish two kind of parts: manufacturing intermediary
    parts which are single items only, and parts ready to be assembled that can
    be wether a single item or pre-assembled items.
    A Part is the lowest level component.

    The list of Item_Parts involved in a product configuration and their
    respective properties, constitute the Bill of Materials of this
    configuration.

    """
    multidmd=Field(Boolean)

    def __init__(self, name, ref, index):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        ref
            Empty documentation.

        index
            Empty documentation.

        """
        # TODO
        Item.__init__(self, name, ref, index)



class ItemAssembly(Item):
    """
    Description
    ===========
    An Item_Assemblyis a gathering of several item_parts or sub-assemblies. An
    Assembly is a composition of its subassemblies and parts.
    """
    has_representation=Field(Boolean)

    def __init__(self, name, ref, index):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        ref
            Empty documentation.

        index
            Empty documentation.

        """
```

```
        # TODO
        Item.__init__(self, name, ref, index)



class StructuralAssembly(DesignArtifact):
    """
    Description
    ===========
    The structural assembly class is necessary to represent and store the
    breakdown elements of a system that defines the hierarchy and the various
    groups of a product structure.
    """
    def __init__(self, name):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        ref
            Empty documentation.

        index
            Empty documentation.


        # TODO
        """
        super(DesignArtifact, self).__init__(name)



class Group(StructuralAssembly):
    """
    Description
    ===========
    Group is a sub-type of Structural -Assembly that permits to gather
    components together. They are mainly used for groups of multi-instantiated
    parts/assemblies.
    """

    def __init__(self, name):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        ref
            Empty documentation.

        index
            Empty documentation.

        """
        # TODO

        super(StructuralAssembly, self).__init__(name)


class Root(StructuralAssembly):
    """
    Description
    ===========
    Empty documentation.

    Relationships
    -------------
    is_root_for
        The product class for which the component is the root.
```

```
    """
    is_root_for = OneToMany('Configuration')



    def __init__(self, name, conf):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        root_for
            Empty documentation.

        """
        # TODO
        super(StructuralAssembly, self).__init__(name)
        self.is_root_for.append(conf)



class Interface(DesignArtifact):
    """
    Description
    ===========
    An Interface defines the physical relationships between product components.

    Physical or theoretical boundaries where two or more system components meet
    and interact and where domain-specific applied rules and conventions define
    their interaction and related design intent.
    These rules and conventions concern domain-specific physical features
    (mechanical, electrical, thermal, etc.) semantic or functional features,
    and potential exchanges of information.


    Relationships
    -------------
    related_comps
        List of components connected by the link.

    mating_features
        List of mating features involved in the interface.

    linkage
        Empty documentation.

    """
    related_comps = ManyToMany('NAUO', inverse='links')
    mating_features = OneToMany('InterfaceTopology')
    linkage = ManyToOne('Interaction')


    def __init__(self, name, linkage):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        nauos
            Empty documentation.

        linkage
            Empty documentation.

        """
        # TODO
        DesignArtifact.__init__(self, name)
        self.linkage=linkage
```

```
    def set_mating_features(self):
        """
        Description
        ===========
        Method to associate a set of mating_features to the interface definition.
        """
        # TODO
        raise NotImplementedError


    def get_topology(self):
        """
        Description
        ===========
        Returns the set of mating features that specify the design intent of the
        interface.
        """
        # TODO
        raise NotImplementedError


class NAUO(Entity):
    """
    Description
    ===========
    Next Assembly Usage Occurence: it represents a single individual occurrence
    of an artefact design definition as used in an immediate next higher parent
    assembly. The name attribute contains a unique instance identifier for the
    individual definition usage occurrence.

    NAUOs permits to:
    - instantiate design artifacts within a product structure
    - localize components in a product structure indicating their respective
    parent components.
    - indicate the definition used by components/instances
    - assign a function (via the attributes sns/sin)
    - configure a product structure applying effectivities on it.


    Fields
    ------
    name
        Empty documentation.

    sns
        Empty documentation.

    sin
        Empty documentation.

    description
        Empty documentation.

    configurable
        Empty documentation.

    domain_related
        Empty documentation.


    Relationships
    -------------
    definition
        Definition used by the instance.



    parent
        Parent assembly definition of the child instance.

    links
        List of links that belongs to the instance/component
```

```
    position_to_parent
       The transformation matrix of the instance.

    props
       List of properties describing the instance.

    linkage
       The interaction ensured by the connector.

    slaves
       List of slaves NAUO dependant of the definition of the master NAUO.

    master
       Empty documentation.

    ports
       List of ports belonging to a component.

    port
       The ports that is delegated on the diffrent component.

    int_topo
       Empty documentation.

    effs
       List of effectivities on which a nauo is relevant to be used.

    change_p
       Empty documentation.

    change_n
       Empty documentation.

    """
    name = Field(String(64))
    sns = Field(String(32))
    sin = Field(String(32))
    description = Field(String(64))
    configurable = Field(Boolean)
    domain_related = Field(Boolean)
    fullname=Field(String(128))


    definition = ManyToOne('DesignDefinition')
    parent = ManyToOne('DesignDefinition')
    links = ManyToMany('Interface')
    position_to_parent = OneToOne('InstancePlacement', inverse='instance')
    props = OneToMany('InstanceProperty', inverse='instance')
    linkage = ManyToOne('Interaction')
    slaves = OneToMany('NAUO', inverse='master')
    master = ManyToOne('NAUO')
    ports = OneToMany('Port')
    int_topo = ManyToOne('InterfaceTopology')
    effs = ManyToMany('Effectivity')
    change_p = ManyToOne('ReplacedUsageRelationship')
    change_n = ManyToOne('ReplacedUsageRelationship')


    def __init__(self, name, sns, definition):
       """
       Description
       ===========
       Construct a new instance of NAUO.

       Parameters
       ----------
       sns
          Empty documentation.

       definition
          Empty documentation.

       configurable
          Empty documentation.
```

```
    effs
        Empty documentation.

    """
    # TODO

    self.name=name
    self.sns=sns
    self.definition=definition



def __repr__(self):
    """
    Description
    ===========
    Return instance's representation: what the method print(nauo) returns.
    """
    # TODO
    return "<NAUO '{}'>".format(self.name)


def get_position(self):
    Position=[]
    #x axis components
    Position.append(self.position_to_parent.rotxx)
    Position.append(self.position_to_parent.rotxy)
    Position.append(self.position_to_parent.rotxz)
    #y axis components
    Position.append(self.position_to_parent.rotyx)
    Position.append(self.position_to_parent.rotyy)
    Position.append(self.position_to_parent.rotyz)
    #z axis components
    Position.append(self.position_to_parent.rotzx)
    Position.append(self.position_to_parent.rotzy)
    Position.append(self.position_to_parent.rotzz)
    #origin point coordinates
    Position.append(self.position_to_parent.trans_x)
    Position.append(self.position_to_parent.trans_y)
    Position.append(self.position_to_parent.trans_z)

    return Position


def load_in_catia(self):
    if "CNEXT.exe" in [psutil.Process(i).name for i in psutil.get_pid_list()]:
        dispatch = win32com.client.dynamic._GetGoodDispatch("CATIA.Application")
        typeinfo = dispatch.GetTypeInfo()
        attr = typeinfo.GetTypeAttr()
        olerepr = win32com.client.build.DispatchItem(typeinfo, attr, None, 0)
        CATIA = win32com.client.dynamic.CDispatch(dispatch, olerepr)
        dispatch = typeinfo = attr = olerepr = None
        #Set the CATIA popup file alerts to False
        #It prevents to stop the macro at each alert during its execution
        CATIA.DisplayFileAlerts = False
        CATIA.RefreshDisplay = True

        prddoc=CATIA.Documents.Add("Product")
        prddoc.Activate
        prdroot=CATIA.ActiveDocument.Product
        prdroot.PartNumber = self.definition.artifact.ref
        prdroot.Revision = self.definition.version
        prdroot.Definition = "J"
        prdroot.Nomenclature=self.sns
        prdroot.DescriptionRef = self.description

        children=self.definition.children
        load_children(prdroot, children)

        specsAndGeomWindow1 = CATIA.ActiveWindow
        viewer3D1 = specsAndGeomWindow1.ActiveViewer
        viewer3D1.Reframe
```

```
def get_mass(self):

   mass = [elt for elt in self.props if isinstance(elt, InstanceMass)][0]
   if mass != None:
      return self.props[2]
   else:
      mass_=0
      for child in self.definition.children:
         if isinstance(child.definition.artifact, ItemPart):
            child_model=[elt for elt in child.definition.models if isinstance(elt, CADModel)][0]
            print child_model.get_mass()
            mass_ += child_model.get_mass()


         mass_ += child.get_mass()
      return mass_

def print_mass(self):
   mass = [elt for elt in self.props if isinstance(elt, InstanceMass)][0]
   if mass != None:
      return "<NAUO '{}'   Mass = {} {}>".format(self.name, mass.mass_value, mass.unit)
   else:
      mass_=0
      for child in self.definition.children:
         if isinstance(child.definition.artifact, ItemPart):
            child_model=[elt for elt in child.definition.models if isinstance(elt, CADModel)][0]
            print child_model.get_mass()
            mass_ += child_model.get_mass()


         mass_ += child.get_mass()
      return mass_



def get_volume(self):
   vol=0.0
   for child in self.definition.children:
      if isinstance(child.definition.artifact, ItemPart):
         child_model=[elt for elt in child.definition.models if isinstance(elt, CADModel)][0]
         vol += child_model.get_volume()


      vol += child.get_volume()

   return vol

def print_volume(self):
   vol=0.0
   for child in self.definition.children:
      if isinstance(child.definition.artifact, ItemPart):
         child_model=[elt for elt in child.definition.models if isinstance(elt, CADModel)][0]
         vol += child_model.get_volume()

      vol += child.get_volume()


   return "<NAUO '{}'   Volume = {} m3>".format(self.name, vol)


def get_inertia(self):
   return self.props[1]


def get_CG(self):
   return self.props[0]


def set_id(self):
   """
   Description
   ===========
```

```
    Empty documentation.
    """
    # TODO
    raise NotImplementedError

  def get_def(self):
    """
    Description
    ===========
    Return instance's design definition.
    """
    # TODO
    raise NotImplementedError

  def get_parent(self):
    """
    Description
    ===========
    Returns the parent assembly design definition.
    """
    # TODO
    raise NotImplementedError

  def add_effectivity(self, conf):
    """
    Description
    ===========
    Add an existing effectivity to the instance effectivity list.
    """
    # TODO
    for eff in self.effs:
      if isinstance(eff, datamodel.ConfManagement.conf.EffectivityList):
        eff.add_conf(conf)
      else:
        print "Only Effectivity_List method is implemented"
        raise NotImplementedError


  def new_effectivity(self):
    """
    Description
    ===========
    Create a new effectivity and add it to the instance's effectivities.
    """
    # TODO
    raise NotImplementedError



  def rem_effectivity(self):
    """
    Description
    ===========
    Remove an effectivity from the instance's effectivity list.
    """
    # TODO
    raise NotImplementedError

  def is_effective(self, confname):
    """
    Description
    ===========
    Return a boolean that describes if the instance is effective for a given
    conf.

    Parameters
    ----------
    conf
      Empty documentation.

    """
    # TODO
    a=0
    conf=datamodel.ConfManagement.conf.Configuration.get_by(name=confname)
```

```
        for eff in self.effs:
            if isinstance(eff, datamodel.ConfManagement.conf.EffectivityList):
                if conf in eff.confs:
                    a=1
                    break
                else:
                    a=0

            #elif eff isinstance(datamodel.ConfManagement.conf.Effectivity, DatedEffectivity):

            elif isinstance(eff, datamodel.ConfManagement.conf.RangeEffectivity):
                if conf.name< eff.end_conf and conf.name>eff.start_conf:
                    a=1
                    break
                else:
                    a=0

        if a==1:
            return True
        else:
            return False


    def multi_instantiate(self):
        """
        Description
        ===========
        create a set  of new nauos using the same design definition under the same
        parent assembly.
        """
        # TODO
        raise NotImplementedError

    def create_slave_nauo(self):
        """
        Description
        ===========
        Construct a new instance of NAUO using the same definition that will follow
        the master nauo changes.
        """
        # TODO
        raise NotImplementedError

    def get_master(self):
        """
        Description
        ===========
        returns the master nauo of a slave nauo.
        """
        # TODO
        raise NotImplementedError

    def placement(self):
        """
        Description
        ===========
        returns the position matrix indicating the position of the nauo in its
        parent assembly coordinate system.
        """
        # TODO
        raise NotImplementedError

    def mass_properties(self):
        """
        Description
        ===========
        returns the mass properties of the instance (centre of mass, inertia) in
        the parent assembly's coordinate system.
        """
        # TODO
        raise NotImplementedError

    def create_block(self):
        """
```

```
        Description
        ===========
        Create a new instance of component block corresponding to the nauo.
        """
        # TODO
        raise NotImplementedError

    def get_ports(self):
        """
        Description
        ===========
        returns the set of ports associated to the nauo.
        """
        # TODO
        raise NotImplementedError

    def add_port(self):
        """
        Description
        ===========
        Create a new port instance to associate to the nauo.
        """
        # TODO
        raise NotImplementedError

    def rem_port(self):
        """
        Description
        ===========
        Delete an existing port associated to the nauo.
        """
        # TODO
        raise NotImplementedError

def load_children(prd, children):

    dispatch = win32com.client.dynamic._GetGoodDispatch("CATIA.Application")
    typeinfo = dispatch.GetTypeInfo()
    attr = typeinfo.GetTypeAttr()
    olerepr = win32com.client.build.DispatchItem(typeinfo, attr, None, 0)
    CATIA = win32com.client.dynamic.CDispatch(dispatch, olerepr)
    CATIA.DisplayFileAlerts = True
    dispatch = typeinfo = attr = olerepr = None

    products=prd.Products
    for child in children:
        if isinstance(child.definition.artifact, ItemAssembly):
            child_catia=products.AddNewProduct(child.definition.artifact.ref)
            child_catia.name=child.name
            child_catia.Nomenclature=child.sns
            child_catia.Revision=child.definition.version
            child_catia.DescriptionRef=child.description

            child_position=child.get_position()
            child_catia.Position.SetComponents (child_position)

            child_children=child.definition.children
            load_children(child_catia, child_children)

        if isinstance(child.definition.artifact, ItemPart):
            model= [elt for elt in child.definition.models if isinstance(elt, CADModel)][0]
            modelpath=model.digitalfile.path
            CATIA.DisplayFileAlerts = True
            print modelpath
            products.AddComponentsFromFiles ([modelpath], "All")
            print "ok"


class DesignDefinition(Entity):
    """
    Description
    ===========
    Design_definition (~ ddid  in AP214) provides the technical definition of
    an artefact within a specific context (view).
```

A technical definition consists of a set of representations (models) and properties (mainly mass and material properties).

For an assembly, the Design_Definition also encompass the composition of its children components. These children components are attached their parent assembly through the use of the NAUO (Next Assembly Usage Occurence) class that represents the nodes of the product structure.

A Design_definition is a characterization of a design artefact definition version, relevant in one or more context application domains and one or more life cycle stages (view_definition_contexts).A design_definition is a collector of the properties that characterize the Product_version in the initial_context and additional_contexts (AP239).

Fields
---
version_id
Empty documentation.

status
Empty documentation.

description
Empty documentation.

has_additional_context
Empty documentation.

creation_date
Empty documentation.

modif_date
Empty documentation.

Relationships
---
nauos
List of occurrences of the instance.

children
List of children instances under the parent assembly.

props
List of properties of the design definition of an artefact.

models
List of models (nominal CAD model, idealized CAD model, CAE model, etc.) that permits to define geometry of the artefact.

artefact
The collector of data common to all versions of an artefact.

successors
List of versions derived from one previous version.

ancestor
The previous version.

change_n
Empty documentation.

change_p
Empty documentation.

additional_contexts
The set of instances of View_definition_context in which this Product_view_definition is also relevant.

initial_context

```
    The View_definition_context in which the defined design definition
    version has been primarily characterized.



  context
    Empty documentation.

  modified_by
    The identification of the user that has modified the definition (change
    of version).

  created_by
    The identification of the user that has created the definition version.

  """
  version = Field(String(4))
  status = Field(String(64))
  description = Field(String(64))
  has_additional_context = Field(Boolean)
  creation_date = Field(Date)
  modif_date = Field(Date)
  nauos = OneToMany('NAUO', inverse='definition')
  children = OneToMany('NAUO', inverse='parent')
  props = OneToMany('Property')
  models = OneToMany('Model')
  artifact = ManyToOne('DesignArtifact')
  successors = OneToMany('DesignDefinition', inverse='ancestor')
  ancestor = ManyToOne('DesignDefinition')
  change_n = ManyToOne('DefinitionVersionRelationship')
  change_p = ManyToOne('DefinitionVersionRelationship')
  additional_contexts = OneToMany('ViewDefinitionContext')
  initial_context = ManyToOne('ViewDefinitionContext', use_alter=True)
  context = ManyToOne('ViewDefinitionContext', use_alter=True)
  modified_by = ManyToOne('User', inverse='modified_defs')
  created_by = ManyToOne('User', inverse='created_defs')

  def __init__(self, version, status, initial_context):
    """
    Description
    ===========
    Construct a new instance of Design Definition (new version)

    Parameters
    ----------
    version
      Empty documentation.

    status
      Empty documentation.

    description
      Empty documentation.

    ivdc
      Empty documentation.

    has_avdc
      Empty documentation.

    """
    # TODO
    self.version=version
    self.status=status
    self.initial_context=initial_context



  def __repr__(self):
    """
    Description
    ===========
    Return instance's representation: what the method print(design_definition)
    returns.
```

```
    """
    # TODO
    raise NotImplementedError

def add_child(self):
    """
    Description
    ===========
    Add an existing NAUO to the related design definition's children.
    """
    # TODO
    raise NotImplementedError

def new_child(self):
    """
    Description
    ===========
    Create a Next Assembly Usage Occurence (NAUO) and add it to the instance's
    children.
    """
    # TODO
    raise NotImplementedError

def rem_child(self):
    """
    Description
    ===========
    Remove a NAUO from the related design definition's children.
    """
    # TODO
    raise NotImplementedError

def new_nauo(self, nauo):
    """
    Description
    ===========
    Create a new NAUO of the definition.
    """
    # TODO
    self.nauo.append(nauo)


def rem_nauo(self):
    """
    Description
    ===========
    Remove a NAUO of the definition.
    """
    # TODO
    raise NotImplementedError

def set_ivdc(self):
    """
    Description
    ===========
    Set the existing intial view definition context in which the definition is
    relevant to use.
    """
    # TODO
    raise NotImplementedError

def add_avdc(self):
    """
    Description
    ===========
    Add an existing additional context in which the definition is relevant to
    use.
    """
    # TODO
    raise NotImplementedError

def new_ivdc(self):
    """
    Description
```

```
    ===========
    Create a new intial context in which the definition is relevant to use.
    """
    # TODO
    raise NotImplementedError

  def new_avdc(self):
    """
    Description
    ===========
    Create a new additional context in which the definition is relevant to use.
    """
    # TODO
    raise NotImplementedError

  def get_contexts(self):
    """
    Description
    ===========
    Return design_defintion's related view defintiion contexts in which the
    definition is used.
    """
    # TODO
    raise NotImplementedError

  def get_models(self):
    """
    Description
    ===========
    Return design_defintion's related models instances.
    """
    # TODO
    raise NotImplementedError

  def add_model(self):
    """
    Description
    ===========
    Add an existing model (present in the models' vault) as part of the
    definition
    """
    # TODO
    raise NotImplementedError

  def new_model(self):
    """
    Description
    ===========
    Create a new model for the definition
    """
    # TODO
    raise NotImplementedError

  def generate_assy_cad(self):
    """
    Description
    ===========
    Method (only applicable for Item_Assemblies) generating an AllCATPart of
    the assembly in CATIA, save it in the CAD files' vault and associate it to
    the CAD model of the assembly.
    """
    # TODO
    raise NotImplementedError

  def rem_model(self):
    """
    Description
    ===========
    Remove a model from the list of models of the definition.
    """
    # TODO
    raise NotImplementedError

  def properties(self):
```

```
        """
        Description
        ===========
        Return design_defintion's related properties.
        """
        # TODO
        raise NotImplementedError

    def set_property(self):
        """
        Description
        ===========
        Permit to create/modify certain item properties of the definition.
        """
        # TODO
        raise NotImplementedError

    def creator(self):
        """
        Description
        ===========
        Return the user who has created the definition.
        """
        # TODO
        raise NotImplementedError

    def modif(self):
        """
        Description
        ===========
        Return the user who has modified the definition.
        """
        # TODO
        raise NotImplementedError

    def suscribers(self):
        """
        Description
        ===========
        Return the list of users that have suscribed to the definition.
        """
        # TODO
        raise NotImplementedError

    def history(self):
        """
        Description
        ===========
        Return all the previous definition versions of one artefact design
        definition version.
        """
        # TODO
        raise NotImplementedError


class Property(Entity):
    """
    Description
    ===========
    A property is an attribute or a characteristic that complete the definition
    of an artifact. these properties are independant from the various instances
    of the artifact.
    Each Property is either an Item property or a Shape dependent property.


    Fields
    ------
    name
        Empty documentation.

    value
        Empty documentation.

    unit
```

```
    Empty documentation.

description
    Empty documentation.


Relationships
-------------
definition
    The Design definition that is described by the related set of
    properties.

change_a
    Empty documentation.

change_d
    Empty documentation.

change_m
    Empty documentation.

value_change_o
    Empty documentation.

value_change_n
    Empty documentation.

"""
name = Field(String(64))
value = Field(Float)
unit = Field(String(64))
description = Field(String(64))
definition = ManyToOne('DesignDefinition', inverse='props')
change_a = ManyToOne('PropertyChange')
change_d = ManyToOne('PropertyChange')
change_m = ManyToOne('PropertyChange')
value_change_o = ManyToOne('ValueChange')
value_change_n = ManyToOne('ValueChange')


def __init__(self, definition, name):
    """
    Description
    ===========
    Construct a new instance of Property.

    Parameters
    ----------
    name
        Empty documentation.

    """
    # TODO
    self.definition=definition
    self.name=name


def __repr__(self):
    """
    Description
    ===========
    Return instance's representation: what the method print(property) returns.
    """
    # TODO
    raise NotImplementedError

def valuate(self):
    """
    Description
    ===========
    Assign a value to the property.
    """
    # TODO
    raise NotImplementedError
```

```
class ShapeDependantProperty(Property):
    """
    Description
    ===========
    A shape_dependant_property is an artifact propertry that is derived from
    its shape (e.g. volume, mass properties) defined in the CAD_Representation.

    Relationships
    -------------
    shape
        The CAD model that provides the shape_dependant_properties.

    """
    shape = ManyToOne('CADModel', inverse='props')

class MassProperty(ShapeDependantProperty):
    """
    Description
    ===========
    Mass properties in the local coordinate system of the artifact. They are
    calculated from the shape and material parameter defined in the CAD model.
    """

class Volume(ShapeDependantProperty):
    """
    Description
    ===========
    Volume of the geometry defined in the CAD model.
    """
    body_v = OneToOne('ShapeBody', inverse='body_volume')


    def __init__(self, value, unit, definition, name='Volume'):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        shape
            Empty documentation.

        """
        # TODO
        ShapeDependantProperty.__init__(self, definition, name)
        self.value=value
        self.unit=unit

    def __repr__(self):
        return "Volume= '%s' '%s'" % (self.value, self.unit)

class CalculatedMass(MassProperty):
    """
    Description
    ===========
    Mass of the artifact.
    """

    body_m=OneToOne('ShapeBody', inverse='body_mass')

    def __init__(self, value, unit, definition, name='Mass'):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        shape
            Empty documentation.
```

```
        """
        # TODO
        MassProperty.__init__(self, definition, name)
        self.value=value
        self.unit=unit


class CgAbs(MassProperty):
    """
    Description
    ===========
    Center of mass in the local coordinate system of the artifact.

    Fields
    ------
    cgx
        Empty documentation.

    cgy
        Empty documentation.

    cgz
        Empty documentation.

    """
    cgx = Field(Float)
    cgy = Field(Float)
    cgz = Field(Float)



    def __init__(self, nauo, definition, name='Local Center of Mass'):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        MassProperty.__init__(self, definition, name)
        #MODIFIER AVEC LES VALEURS RELATIVES

        self.cgx=CgRel.get_by(instance=nauo).cgx+nauo.position_to_parent.trans_x
        self.cgy=CgRel.get_by(instance=nauo).cgy+nauo.position_to_parent.trans_y
        self.cgz=CgRel.get_by(instance=nauo).cgz+nauo.position_to_parent.trans_z


class InertiaAbs(MassProperty):
    """
    Description
    ===========
    Moment of Inertia calculated in the local coordinate system of the artifact.

    Fields
    ------
    polar_x
        Empty documentation.

    diametral_y
        Empty documentation.

    diametral_z
        Empty documentation.

    """
    polar_x = Field(Float)
    diametral_y = Field(Float)
    diametral_z = Field(Float)


    def __init__(self, nauo, definition, name='Local Inertia Moments'):
        """
        Description
        ===========
        Empty documentation.
```

```
        """
        # TODO
        MassProperty.__init__(self, definition, name)
        #MODIFIER AVEC LES VALEURS RELATIVES

        rx=[nauo.position_to_parent.rotxx, nauo.position_to_parent.rotxy,
            nauo.position_to_parent.rotxz]
        ry=[nauo.position_to_parent.rotyx, nauo.position_to_parent.rotyy,
            nauo.position_to_parent.rotyz]
        rz=[nauo.position_to_parent.rotzx, nauo.position_to_parent.rotzy,
            nauo.position_to_parent.rotzz]
        RotMatrix=np.array([rx, ry, rz])
        InvRotMatrix=np.linalg.inv(RotMatrix)
        Ix=[nauo.props[1].polar_x, 0, 0]
        Iy=[0, nauo.props[1].diametral_y, 0]
        Iz= [0, 0, nauo.props[1].diametral_z]
        InertiaMatRel=np.array([Ix, Iy,Iz])
        B=np.dot(RotMatrix, InertiaMatRel)
        InertiaMatAbs=np.dot(B, InvRotMatrix)

        self.polar_x=InertiaMatAbs.item(0)
        self.diametral_y=InertiaMatAbs.item(4)
        self.diametral_z=InertiaMatAbs.item(8)


class ItemProperty(Property):
    """
    Description
    ===========
    An Item_Property is a property that is totally independant from the
    geometry of the artifact (e.g. Material information and properties).
    """


class PartProperty(ItemProperty):
    """
    Description
    ===========
    Properties that can be assigned only on leaf components (Item_Part).
    """


class Material(Property):
    """
    Description
    ===========
    Material reference (DMD) parameter defined in the CAD model.

    Fields
    ------
    dmd
        Empty documentation.


    Relationships
    -------------
    body
        Empty documentation.

    material
        The material assigned to the Item_Part property.

    density
        The density/volumic mass corresponding the material defined in the CAD
        model.

    """
    refdmd = Field(String(16))
    body = OneToOne('ShapeBody', inverse='material')
    props = OneToMany('MaterialProperty', inverse='mat')



    def __init__(self, dmd, definition, name='Material'):
```

```
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        dmd
            Empty documentation.

        """
        # TODO
        Property.__init__(self, definition, name)
        self.refdmd=dmd


    def __repr__(self):
        return "<'%s' :  DMD='%s'" % (self.name, self.refdmd)



    def get_dmd_bom(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        raise NotImplementedError


class MaterialProperty(Property):
    """
    Description
    ===========
    Material physical property.

    Relationships
    -------------
    mat
        The material described by the properties.

    """
    mat = ManyToOne('Material')


    def __init__(self, mat, definition, name):
        Property.__init__(self, definition, name)
        self.mat=mat


class Density(MaterialProperty):
    """
    Description
    ===========
    Denisty or volumic mass of the material.

    """



    def __init__(self, mat, value, unit, definition, name='Volumic Mass'):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        mat
            Empty documentation.

        """
        # TODO
        MaterialProperty.__init__(self, mat, definition, name)
```

```
        self.value=value
        self.unit=unit


    def __repr__(self):
        return "'%s' :  '%s' '%s'" %(self.name, self.value, self.unit)



class Model(Entity):
    """
    Description
    ===========
    A model is a geometric representation of an artifact. In this case we only
    deals with CAD models that geometrically represent the more or less exact
    shape of an artifact.

    Fields
    ------
    name
        Empty documentation.

    model_type
        Empty documentation.

    revision_id
        Empty documentation.


    Relationships
    -------------
    definition
        Definition that is describe by the model.

    files
        List of files where the model is stored.

    derived_from
        The previous model revision.

    derived_models
        List of model revisions derived from one previous model revision.

    change_n
        Empty documentation.

    change_p
        Empty documentation.

    """
    name = Field(String(64))
    revision_id = Field(String(4))
    definition = ManyToOne('DesignDefinition', inverse='models')
    digitalfile = OneToOne('DigitalFile', inverse='model')
    derived_from = ManyToOne('Model', inverse='derived_models')
    derived_models = OneToMany('Model')
    change_n = ManyToOne('ModelRevisionRelationship')
    change_p = ManyToOne('ModelRevisionRelationship')



    def __init__(self, name, revision_id, digitalfile):
        """
        Description
        ===========
        Construct a new instance of Model.

        Parameters
        ----------
        file
            Empty documentation.

        name
            Empty documentation.
```

```
        type
            Empty documentation.

        rev
            Empty documentation.

        """
        # TODO
        self.name=name
        self.revision_id=revision_id
        self.digitalfile=digitalfile


    def __repr__(self):
        """
        Description
        ===========
        Return instance's representation: what the method print(model) returns.
        """
        # TODO
        return "<Model  name:'%s';  revision:'%s'>" %(self.name, self.revision_id)


    def new_revision(self, name, previous_revision_id, newdigitalfile):
        self.__init__(self, name, previous_revision_id+1, newdigitalfile)
        self.derived_from=self.get_by(name=self.name, revision_id=previous_revision_id)


    def history(self):
        """
        Description
        ===========
        Return all the previous model revisions of one model revision.
        """
        # TODO
        previous_model=self.derived_from

        return previous_model

        self.history(previous_model)



class CADModel(Model):
    """
    Description
    ===========
    CAD representation that  defines the nominal shape of the artifact design
    definition.

    Fields
    ------
    rep_type
        Empty documentation.


    Relationships
    -------------
    property
        List of shape_dependant_properties derived from the CAD model.

    bodies
        List of bodies that compose the CAD model.

    publis
        List of mating feature publications present in the model.

    """
    rep_type = Field(String(64))
    props = OneToMany('ShapeDependantProperty')
    bodies = OneToMany('ShapeBody')
    publis = OneToMany('MatingFeaturePublication')
```

```
def __init__(self, name, revision_id, digitalfile):
    Model.__init__(self, name, revision_id, digitalfile)


def __repr__(self):
    return "<CADModel  name:'%s';  rev:'%s'>" %(self.name, self.revision_id)



def get_bodies(self):
    """
    Description
    ===========
    Return the
    """
    # TODO
    for body in self.bodies:
        body.__repr__()

    return self.bodies


def mating_features(self):
    """
    Description
    ===========
    Returns the list of meating feature publications defined in the CAD model.
    """
    # TODO
    return self.publis


def get_volume(self):
    volpart=0
    for vol in [elt for elt in self.props if isinstance(elt, Volume)]:
        volpart += vol.value
        #vol_unit=vol.unit
    #print "<CADModel: '%s'   Volume= '%f' '%s'>" %(self.name, volpart, vol_unit)
    return volpart

def print_volume(self):
    volpart=0
    for vol in [elt for elt in self.props if isinstance(elt, Volume)]:
        volpart += vol.value
        vol_unit=vol.unit
    print "<CADModel: '%s'   Volume= '%f' '%s'>" %(self.name, volpart, vol_unit)


def get_mass(self):
    masspart=0
    for mass in [elt for elt in self.props if isinstance(elt, CalculatedMass)]:
        masspart += mass.value
        #massunit=mass.unit
    #print "<CADModel: '%s'   Mass= '%s' '%s'>" %(self.name, masspart, massunit)
    return masspart

def print_mass(self):
    masspart=0
    for mass in [elt for elt in self.props if isinstance(elt, CalculatedMass)]:
        masspart += mass.value
        massunit=mass.unit
    print "<CADModel: '%s'   Mass= '%s' '%s'>" %(self.name, masspart, massunit)


def get_dmds(self):
    """
    Description
    ===========
    Retrieve the dmd property for each shape body that compose the CAD model.

    Parameters
    ----------
    file
        Empty documentation.
```

```
        """
        # TODO
        for body in self.bodies:
            if body.dmd.refdmd== None:
                return "Body_Name: '%s'  DMD: Not sepcified" %(body.name)
            else:
                return "Body_Name: '%s'  DMD:'%s'" %(body.name, body.dmd.refdmd)


    def get_mass_prop(self):
        """
        Description
        ===========
        Retrieve the mass properties of the CAD model in the local coordinate system
        of the part/assembly.
        """
        # TODO
        cg=[elt for elt in self.props if isinstance(elt, CgAbs)]
        inertia=[elt for elt in self.props if isinstance(elt, InertiaAbs)]
        model_mass=self.get_mass()

        print  "<Model_Name:  '%s'  Mass_Properties(mass='%f';CGLocal=['%f';  '%f';  '%f'];  InertiaLocal=['%f';  '%f';  '%f'])>"  %(self.name,
model_mass, cg[0], cg[1], cg[2], inertia[0], inertia[1], inertia[2])
        return model_mass
        return cg
        return inertia


    def open_model(self):
        import psutil
        if "CNEXT.exe" in [psutil.Process(i).name for i in psutil.get_pid_list()]:
            import win32com.client
            CATIA = win32com.client.Dispatch("CATIA.Application")
            CATIA.RefreshDisplay = True
            CATIA.Documents.Open(self.digitalfile.path)

            #Set the CATIA popup file alerts to False
            #It prevents to stop the macro at each alert during its execution
            CATIA.DisplayFileAlerts = False
            CATIA.RefreshDisplay = True

            opened_model= CATIA.ActiveDocument

            return opened_model
        else:
            print "CATIA is not running. Please open a CATIA application"


class DigitalFile(Entity):
    """
    Description
    ===========
    Interface to work with vault's CAD files.
    Theses digital_files permit to open/read the CAD file in a CAD application
    (e.g CATIA, NX, etc.).


    Fields
    ------
    name
        Empty documentation.

    format
        Empty documentation.


    Relationships
    -------------
    model
        Model which the file correspond to.

    template
        Empty documentation.
```

```
    """
    path = Field(String(128))
    format_ = Field(String(64))
    model = ManyToOne('Model')
    template = ManyToOne('JointTemplate')



    def __init__(self, path):
        """
        Description
        ===========
        Create a new ExternalFile instance.

        Parameters
        ----------
        path
            Path of the file to load.

        """
        # TODO
        self.path=path

    def openfile(self, appli_path):
        """
        Description
        ===========
        Return a python read-only file object that is represented by the instance.
        """
        # TODO
        import subprocess
        subprocess.Popen("%s %s" % (appli_path, self.path))

        #open(self.path, 'r+b')

    def set(self):
        """
        Description
        ===========
        Load a new file and associate it to the instance
        """
        # TODO
        raise NotImplementedError


class Interaction(Entity):
    """
    Description
    ===========
    Interactions are the physical phenomena that occur at the interfaces
    between connected components.
    Their definition specify the domain-specific physical,functional and
    semantic features that define the rules and conventions to apply.


    Fields
    ------
    domain
        Empty documentation.

    direct
        Empty documentation.

    permanent
        Empty documentation.


    Relationships
    -------------
    connectors
        The connector that permits to ensure the interaction.

    interface
```

```
        Empty documentation.

    """
    domain = Field(String(64))
    direct = Field(Boolean)
    permanent = Field(Boolean)
    connectors = OneToMany('NAUO', inverse='linkage')
    interface = OneToOne('Interface', inverse='linkage')

    def __init__(self, direct, permanent):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        interface
            Empty documentation.

        description
            Empty documentation.

        direct
            Empty documentation.

        permanent
            Empty documentation.

        """
        # TODO
        self.direct=direct
        self.permanent=permanent

    def __repr__(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        return "<Interaction (Interface_Name= {};  Domain= {}; Direct={};  Permanent:{})>" .format(self.interface.name, self.domain, self.di-
rect, self.permanent)


    def set_connector(self, nauo):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        nauo
            Empty documentation.

        """
        # TODO
        raise NotImplementedError

    def link_ports(self, ports):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        ports
            Empty documentation.

        """
        # TODO
        raise NotImplementedError
```

```
class SingleDomainInteraction(Interaction):
    """
    Description
    ===========
    Interaction where domain-specific physical phenomena occur.
    """

    def __init__(self, direct, permanent):
        Interaction.__init__(self, direct, permanent)


class CrossDomainInteraction(Interaction):
    """
    Description
    ===========
    Interaction where multiple physical phenomena from multiple
    domains/disciplines occurs.
    """
    def __init__(self, direct, permanent):
        Interaction.__init__(self, direct, permanent)


class UnintendedInteraction(Interaction):
    """
    Description
    ===========
    Interaction (single or cross-domain) that can occur but which is unintended
    and not functionnal. But they need to be specified and characterized to
    anticipate them.
    """

    def __init__(self, direct, permanent):
        Interaction.__init__(self, direct, permanent)


class FluidStructureLinkage(CrossDomainInteraction):
    """
    Description
    ===========
    Interaction generated by the action an internal or surrounding fluid flow
    on some movable or deformable structures.
    """

    def __init__(self, direct, permanent):
        CrossDomainInteraction.__init__(self, direct, permanent)


class MechanicalJoint(SingleDomainInteraction):
    """
    Description
    ===========
    A mechanical linkage is an assembly of bodies connected to manage forces
    and movement. The movement of a body, or link, is studied using geometry so
    the link is considered to be rigid. Therefore a mechanical linkage is
    specified by its kinematic feaures (dof) and its interface topology
    (provided here by CAD mating features which the type and function can be
    captured in templates).

    Fields
    ------
    rigid
        Empty documentation.

    can_be_dismantled
        Empty documentation.


    Relationships
    -------------
    set_of_dof
        Empty documentation.
```

```
    """
    rigid = Field(Boolean)
    can_be_dismantled = Field(Boolean)
    set_of_dof = OneToOne('Dof', inverse='linkage')



    def __init__(self, direct, permanent, rigid=None, can_be_dismantled=None):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        rigid
            Empty documentation.

        can_be_dismantled
            Empty documentation.

        """
        # TODO
        SingleDomainInteraction.__init__(self, direct, permanent)
        self.rigid=rigid
        self.can_be_dismantled=can_be_dismantled


    def set_dof(self):
        """
        Description
        ===========
        Set the values of the associated dof table defining the kinematic of the
        joint.
        """
        # TODO
        raise NotImplementedError

class PreDefinedJoint(MechanicalJoint):
    """
    Description
    ===========
    Usual mechanical joints which the design can be parametrized with templates
    defining a list of specific mating features to design the joint.
    """



    def __init__(self, direct, permanent, rigid, can_be_dismantled):
        MechanicalJoint.__init__(self, direct, permanent, rigid,
                                 can_be_dismantled)


class BearingJoint(PreDefinedJoint):
    """
    Description
    ===========
    A revolute joint made with the use of a support, guide, or locating piece
    for a rotating or reciprocating mechanical part (ball bearing, roll
    bearing).

    Fields
    ------
    technology
        Empty documentation.


    Relationships
    -------------
    trans_blocking
        Empty documentation.

    contact_rotor_stator
        Empty documentation.
```

```
    clearances
      Empty documentation.

    """
    technology = Field(String(64))
    trans_blockings = OneToMany('ShapeContact', inverse='joint_trans')
    contact_rotor_stators = OneToMany('ShapeContact', inverse='joint_rs')
    clearances = OneToMany('FunctionalClearance', inverse='bearing_joint')



    def __init__(self, direct, permanent, can_be_dismantled, techno,
                 rigid=False):
      """
      Description
      ===========
      Construct a new instance of a bearing joint.

      Parameters
      ----------
      techno
        Empty documentation.

      """
      # TODO
      PreDefinedJoint.__init__(self, direct, permanent, rigid,
                               can_be_dismantled)
      self.technology=techno


class GenericJoint(MechanicalJoint):
    """
    Description
    ===========
    Mechanical joints without any pre-defined templates of mating features
    pointers. But the user (e.g. method engineer) can define himself a template
    if the joint need to be re-instanciated.

    Fields
    ------
    type_gen
      Empty documentation.


    Relationships
    -------------
    template
      The template associated to the joint.

    contacts
      Empty documentation.

    clearances
      Empty documentation.

    interferences
      Empty documentation.

    """
    joint_type = Field(String(64))
    template = OneToOne('JointTemplate', inverse='joint')
    contacts = OneToMany('ShapeContact', inverse='generic_joint_co')
    clearances = OneToMany('FunctionalClearance')
    interferences = OneToMany('FunctionalInterference')



    def __init__(self, direct, permanent, rigid=None, can_be_dismantled=None,
                 joint_type=None):
      """
      Description
      ===========
      Construct a new instance of generic joint.
```

```
        Parameters
        ----------
        type
            Empty documentation.

        """
        # TODO
        MechanicalJoint.__init__(self, direct, permanent, rigid,
                                 can_be_dismantled)
        self.joint_type=joint_type



    def generate_template(self):
        """
        Description
        ===========
        Generate a xml file capturing the set of pre-defined named mating features
        that specifies the design intent of the interface whre the joint takes
        place.
        """
        # TODO
        raise NotImplementedError



    def use_template(self, template):
        """
        Description
        ===========
        Returns from a template a set of pre-defined named mating features that
        need to be specify to define the joint.

        Parameters
        ----------
        template
            Empty documentation.

        """
        # TODO
        raise NotImplementedError



class BoltedJoint(PreDefinedJoint):
    """
    Description
    ===========
    Mechanism / assembly made with bolts. They consist of fasteners that
    capture and join other parts, and are secured with the mating of screw
    threads.

    Fields
    ------
    type_bj
        The type of bolted joint : bolted flange, pin joint, etc.


    Relationships
    -------------
    bf_planar
        Empty documentation.

    bolt_nut
        Empty documentation.

    bolt_comp_acl
        Empty documentation.

    bolt_comp_b
        Empty documentation.

    bolt_comp_aco
        Empty documentation.
```

```
    bolt_comps
        Empty documentation.

    other_planars
        Empty documentation.

    """
    type_bj = Field(String(64))
    bolt_nut = OneToOne('ShapeContact', inverse='bolted_joint_n')
    bolt_comp_acl = OneToOne('FunctionalClearance', inverse='bolted_joint_acl')
    bolt_comp_b = OneToOne('FunctionalClearance', inverse='bolted_joint_b')
    bolt_comp_aco = OneToOne('ShapeContact', inverse='bolted_joint_aco')
    bolt_comps = OneToMany('FunctionalClearance', inverse='bolted_joint_s')
    bf_planars = OneToMany('ShapeContact', inverse='bolted_joint_aps')



    def __init__(self, type_bj, direct=False,
                 permanent=True, rigid=True, can_be_dismantled=True):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        type_bj
            Empty documentation.

        """
        # TODO
        PreDefinedJoint.__init__(self, direct, permanent, rigid,
                                 can_be_dismantled)
        self.type_bj=type_bj


    def customize(self):
        """
        Description
        ===========
        Method that extends the defintion of a bolted_flange template when the
        bolted_flange connects more than two components together. In that case,
        there are potential additional clearances, alignment contacts and other
        planar contacts to define.
        The number of mating features to define is directly dependant of the number
        of connected components:

        - 3 components    --> 2 planar contacts
                --> 3 clerances with the bolt
                --> 0 to 2 alignments
        - 4 components    -->3 planar contacts
                --> 4 clearances with the bolt
                --> 0 to 3 alignments

        """
        # TODO
        raise NotImplementedError


class BoltedFlange(BoltedJoint):
    """
    Description
    ===========
    Specific bolted connection for locating, strengthening and connceting two
    cylindrical/rotational parts. The conncetion is made through a radially
    projecting collar or rim on the two connected components and a ring of
    bolts on the circonference.

    Fields
    ------
    aligned
        Empty documentation.
```

```
    nb_holes
        Empty documentation.


    Relationships
    -------------
    alignment
        Empty documentation.

    other_alignments
        Empty documentation.

    """
    aligned = Field(Boolean)
    nb_holes = Field(Integer)
    nb_comps= Field(Integer)
    alignments = OneToMany('ShapeContact', inverse='bolted_joint_als')



    def __init__(self, nb_comps, alignment, nb_holes, type_bj='Bolted Flange',
                 direct=False, permanent=True, rigid=True,
                 can_be_dismantled=True):
        """
        Description
        ===========
        Construct a new instance of a bolted flange.

        Parameters
        ----------
        alignment
            Empty documentation.

        nb_holes
            Empty documentation.

        """
        # TODO
        BoltedJoint.__init__(self, type_bj, direct, permanent, rigid,
                             can_be_dismantled)
        self.nb_comps=nb_comps
        self.alignment=alignment
        self.nb_holes=nb_holes



class Dof(Entity):
    """
    Description
    ===========
    Set of degrees of freedom that defines the kinematic of the mechanical
    joint.

    Fields
    ------
    Tx
        Empty documentation.

    Ty
        Empty documentation.

    Tz
        Empty documentation.

    Rx
        Empty documentation.

    Ry
        Empty documentation.

    Rz
        Empty documentation.
```

```
    Relationships
    -------------
    linkage
      Empty documentation.

    """
    Tx = Field(Boolean)
    Ty = Field(Boolean)
    Tz = Field(Boolean)
    Rx = Field(Boolean)
    Ry = Field(Boolean)
    Rz = Field(Boolean)
    linkage = ManyToOne('MechanicalJoint')



    def __init__(self, linkage):
      """
      Description
      ===========
      Empty documentation.

      Parameters
      ----------
      Tx
        Empty documentation.

      Ty
        Empty documentation.

      Tz
        Empty documentation.

      Rx
        Empty documentation.

      Ry
        Empty documentation.

      Rz
        Empty documentation.

      """
      # TODO
      self.linkage=linkage



    def __repr__(self):
      """
      Description
      ===========
      Empty documentation.
      """
      # TODO
      raise NotImplementedError




class JointTemplate(Entity):
    """
    Description
    ===========
    A generic joint's parametrizes the joint with a set of specific named
    mating_features that need to be specified to define and ensure the joint.
    It allows the re-use/re-instantiation of the joint.
    It consist to associate to the joint an xml file capturing the set of
    pre-defined named mating features todefine.


    Fields
    ------
    name
```

```
    Empty documentation.


  Relationships
  -------------
  joint
    The joint that the template parametrizes.

  file
    Empty documentation.

  """
  name = Field(String(64))
  joint = ManyToOne('GenericJoint')
  file = OneToOne('DigitalFile', inverse='template')



  def __init__(self, name, file):
    """
    Description
    ===========
    Construct a new instance of a joint template.

    Parameters
    ----------
    name
      Empty documentation.

    file
      Empty documentation.

    """
    # TODO
    raise NotImplementedError


class InterfaceTopology(Entity):
  """
  Description
  ===========
  The topology of an interface specifies the area of interaction or
  non-interaction (clearance) that need to be defined between two components.
  It represents a physical relation between two ports or mating features that
  are parts of the two connected components.

  Fields
  ------
  name
    Empty documentation.

  type
    Empty documentation.

  value
    Empty documentation.

  unit
    Empty documentation.


  Relationships
  -------------
  mating_features
    The 2 ports interfacing.

  interface
    The interface in which the mating features are involved.

  ports
    Empty documentation.

  related_comps
    Empty documentation.
```

```
    """
    name = Field(String(64))
    value = Field(Float)
    unit = Field(String(8))
    mating_features = OneToMany('MatingFeaturePublication')
    interface = ManyToOne('Interface', inverse='mating_features')
    ports = OneToMany('Port', inverse='topology')
    mating_comps = OneToMany('NAUO', inverse='int_topo')



    def __init__(self, name, nauo1, nauo2):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        name
            Empty documentation.

        type
            Empty documentation.

        """
        # TODO
        self.name=name
        self.mating_comps.extend([nauo1, nauo2])


    def __repr__(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        return "<Interface Topology  name: '%s';  Comp1: '%s';  Comp2: '%s'>" %(self.name, self.mating_comps[0], self.mating_comps[1])


class FunctionalClearance(InterfaceTopology):
    """
    Description
    ===========
    A Functional_Clearance is an intended empty space between the shapes of two
    components. The clearance is defined as the loosest fit or maximum intended
    spatial distance between mating parts.

    The best example are the functional clearances between a rotor and stator
    elements.


    Relationships
    -------------
    bolted_joint_acl
        Empty documentation.

    bolted_joint_b
        Empty documentation.

    bolted_joint_s
        Empty documentation.

    generic_joint_cl
        Empty documentation.

    bearing_joint
        Empty documentation.

    """
    bolted_joint_acl = ManyToOne('BoltedJoint')
    bolted_joint_b = ManyToOne('BoltedJoint')
```

```
    bolted_joint_s = ManyToOne('BoltedJoint')
    generic_joint_cl = ManyToOne('GenericJoint', inverse='clearances')
    bearing_joint = ManyToOne('BearingJoint')



    def __init__(self, name, nauo1, nauo2):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        value
            Empty documentation.

        unit
            Empty documentation.

        """
        # TODO
        InterfaceTopology.__init__(self, name, nauo1, nauo2)


class ShapeContact(InterfaceTopology):
    """
    Description
    ===========
    A Shape_Contact is a direct intended contact between the shapes of two
    components.

    Relationships
    -------------
    bolted_joint_ap
        Empty documentation.

    bolted_joint_n
        Empty documentation.

    bolted_joint_al
        Empty documentation.

    bolted_joint_aco
        Empty documentation.

    bolted_joint_aps
        Empty documentation.

    bolted_joint_als
        Empty documentation.

    generic_joint_co
        Empty documentation.

    joint_trans
        Empty documentation.

    joint_rs
        Empty documentation.

    """
    bolted_joint_n = ManyToOne('BoltedJoint')
    bolted_joint_aco = ManyToOne('BoltedJoint')
    bolted_joint_aps = ManyToOne('BoltedJoint')
    bolted_joint_als = ManyToOne('BoltedFlange')
    generic_joint_co = ManyToOne('GenericJoint')
    joint_trans = ManyToOne('BearingJoint')
    joint_rs = ManyToOne('BearingJoint')


    def __init__(self, name, nauo1, nauo2):
        InterfaceTopology.__init__(self, name, nauo1, nauo2)
        InterfaceTopology.value=0
```

```
        InterfaceTopology.unit='mm'




class FunctionalInterference(InterfaceTopology):
    """
    Description
    ===========
    A Functional_interference is an intended clash that occurs between shapes
    of two components. In a DMU, this generally occurs for the
    interference/press or friction fits (e.g. interference fit thread, press
    fitting of shafts into bearings or bearings into their housings or
    interference between the abradable coating on the fan case and the fan
    blades).

    Relationships
    -------------
    joint_int
        Empty documentation.

    """
    joint_int = ManyToOne('GenericJoint', inverse='interferences')


    def __init__(self, name, nauo1, nauo2):
        InterfaceTopology.__init__(self, name, nauo1, nauo2)




class FluidSolidBoundary(InterfaceTopology):
    """
    Description
    ===========
    A Fluid_Solid interface is the place where an interaction occurs between
    some movable or deformable structure and an internal or surrounding fluid
    flow.

    A fluid_solid interfaces are located on the structure and generally
    represent the boundaries of a fluid domain.

    """


class MatingFeaturePublication(Entity):
    """
    Description
    ===========
    Mating feature publication are publication of part/portion of the shape of
    a model (generally faces), that are publshed in order to identify and
    localize the interfaces (area of interaction) with other components.

    Fields
    ------
    name
        Empty documentation.


    Relationships
    -------------
    model
        The model that contains the mating feature publications.

    interface
        The interface defined by the two ports.

    port
        The port of the leaf component (item_part instance).

    """
    name = Field(String(64))
    model = ManyToOne('CADModel', inverse='publis')
    interface = ManyToOne('InterfaceTopology', inverse='mating_features')
```

```
    port = ManyToOne('Port')



    def __init__(self, model, name=None):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        name
            Empty documentation.

        """
        # TODO
        self.name=name
        self.model=model


class Port(Entity):
    """
    Description
    ===========
    A port represents an intended interaction between a component
    and its environment. All interactions between components are mediated by
    ports. It specifies the interaction area between two components.


    Relationships
    -------------
    mating_feature
        The name of the publication of the mating feature (face) specified in
        the CAD model.

    nauo
        The component to which the ports belong to.

    delegations
        List of components on which the port is delegated.

    topology
        Empty documentation.

    """
    mating_feature = OneToOne('MatingFeaturePublication', inverse='port')
    nauo = ManyToOne('NAUO', inverse='ports')
    delegations = OneToMany('Port', inverse='port')
    port = ManyToOne ('Port')
    topology = ManyToOne('InterfaceTopology')



    def __init__(self, nauo):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        nauo
            Empty documentation.

        """
        # TODO
        raise NotImplementedError

    def delegate(self, nauo):
        """
        Description
        ===========
        Empty documentation.
```

```
        Parameters
        ----------
        nauo
            Empty documentation.

        """
        # TODO
        raise NotImplementedError

    def remove(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        raise NotImplementedError

    def set_mating_feature(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        raise NotImplementedError

    def linkages(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        raise NotImplementedError



class ShapeBody(Entity):
    """
    Description
    ===========
    A shape body that is part of the CAD model.

    Fields
    ------
    name
        Empty documentation.

    main
        Empty documentation.


    Relationships
    -------------
    model
        The CAD model that encompass the shape bodies.

    dmd
        Empty documentation.

    """
    name = Field(String(64))
    main = Field(Boolean)
    model = ManyToOne('CADModel', inverse='bodies')
    material = ManyToOne('Material')
    body_volume=ManyToOne('Volume')
    body_mass=ManyToOne('CalculatedMass')


    def __init__(self, model, name, main):
        """
        Description
```

```
        ===========
        Empty documentation.

        Parameters
        ----------
        model
            Empty documentation.

        name
            Empty documentation.

        main
            Empty documentation.

        """
        # TODO
        self.model=model
        self.name=name
        self.main=main


    def __repr__(self):
        return "<Body from Model {} :  name={}>".format(self.model.name, self.name)


    def get_dmd(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        return "<BODY name={}  DMD={} >".format(self.name, self.material.refdmd)


class InstancePlacement(Entity):
    """
    Description
    ===========
    An Instance_placement is the information pertaining to the
    placement of a component relatively to the the cartesian coordinate system
    of its parent assembly.

    It corresponds to the transformation matrix that permits to pass from the
    placement of the shape of the corresponding artifact definition in its own
    coordinate system to the intance_placement defined in the coodinate syystem
    of its parent assembly.


    Fields
    ------
    trans_x
        Empty documentation.

    trans_y
        Empty documentation.

    trans_z
        Empty documentation.

    rot_x
        Empty documentation.

    rot_y
        Empty documentation.

    rot_z
        Empty documentation.


    Relationships
    -------------
    instance
        The instance that is positioned by the instance_palcement matrix.
```

```
    """
    trans_x = Field(Float)
    trans_y = Field(Float)
    trans_z = Field(Float)
    rotxx = Field(Float)
    rotxy = Field(Float)
    rotxz = Field(Float)
    rotyx = Field(Float)
    rotyy = Field(Float)
    rotyz = Field(Float)
    rotzx = Field(Float)
    rotzy = Field(Float)
    rotzz = Field(Float)
    instance = ManyToOne('NAUO')



    def __init__(self, nauo):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        nauo
            Empty documentation.

        """
        # TODO
        self.instance=nauo


    def __repr__(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        raise NotImplementedError


    def rot_vect(self, rotxx,rotyx,rotzx, rotzy,rotzz):

        rx=math.atan2(rotzz, rotzy)
        ry=-(math.asin(rotzx))
        rz=math.atan2(rotxx, rotyx)

        return "['%s';  '%s';  '%s']" % (rx, ry ,rz)
        '''
        #convert radians in degrees
        position_entity.rot_x=math.degrees(rx)
        position_entity.rot_y=math.degrees(-ry)
        position_entity.rot_z=math.degrees(rz)
        '''

class InstanceProperty(Entity):
    """
    Description
    ===========
    instance_Property are the sepcific properties relative to an instance
    (mainly relative mass properties). The intance properties values relative
    to the parent component coordonate system.


    Relationships
    -------------
    instance
        Instance that is described by the property.

    """
    name=Field(String(32))
```

```
    instance = ManyToOne('NAUO')
    unit=Field(String(8))

    def __init__(self, nauo, name):
        """
        Description
        ===========
        Empty documentation.

        Parameters
        ----------
        nauo
            Empty documentation.

        position_to_parent
            Empty documentation.

        """
        # TODO
        self.name=name
        self.instance=nauo

    def __repr__(self):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        return "<InstanceProperty '{}'>".format(self.name)

class InstanceMass(InstanceProperty):

    mass_value=Field(Float)


    def __init__(self, mass_value, nauo, name='Mass'):
        InstanceProperty.__init__(self, nauo, name)
        self.mass_value=mass_value

    def __repr__(self):
        return "<Instance: name='{}';  {} = {} {}>".format(self.instance.name, self.name, self.mass_value, self.unit)

class CgRel(InstanceProperty):
    """
    Description
    ===========
    Center of mass in the relative coordinate system of the parent assembly.

    Fields
    ------
    cgx
        Empty documentation.

    cgy
        Empty documentation.

    cgz
        Empty documentation.

    """
    cgx = Field(Float)
    cgy = Field(Float)
    cgz = Field(Float)

    def __init__(self, nauo, name='Center of Mass'):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        InstanceProperty.__init__(self, nauo, name)
```

```
    def __repr__(self):

        return "'%s':  (%f ; %f ; %f) %s" % (self.name, self.cgx, self.cgy, self.cgz, self.unit)

class InertiaRel(InstanceProperty):
    """
    Description
    ===========
    Moment of Inertia calculated in the relative coordinate system of the
    parent assembly.

    Fields
    ------
    polar_x
        Empty documentation.

    diametral_y
        Empty documentation.

    diametral_z
        Empty documentation.

    """
    polar_x = Field(Float)
    diametral_y = Field(Float)
    diametral_z = Field(Float)

    def __init__(self, nauo, name='Inertia Moments'):
        """
        Description
        ===========
        Empty documentation.
        """
        # TODO
        InstanceProperty.__init__(self, nauo,name)

    def __repr__(self):

        return "'%s':  (%f ; %f ; %f) %s" % (self.name, self.polar_x, self.diametral_y, self.diametral_z, self.unit)
'''
appeller une fontion 'get_mass' de NAUO à définir
class InstanceMass(InstanceProperty):

    mass_value=Field(Float)

    def __init__(self, nauo, name='Mass'):
        designdef=nauo.definition
        mass=0
        for child in designdef.children:
            childdef=child.definition
            mass_child=childdef.props.get_by(name='Mass')
            mass=mass+mass_child

        self.mass_value=mass

    def __repr__(self):
    return "<Instance: name='{}'  '{}'='{}' '{}'>".format(self.instance.name, self.name, self.mass_value, self.unit)

'''
class FluidDomain(DesignArtifact):
    """
    Description
    ===========
    Fluid_Element class is another type of system elements, that need to be
    consiedered in the defintion of the product, but that are not tangible for
    the customer or for the manufacturing process; as a result the must be
    integrated in product structures used in specific engineering domains, but
    they do not make part of the Bill of Materials.

    Their defintion (models and properties) is essential for the defintion of
    turbo-machines like an aero-engine.

    """
```

Table 19: Generated Python source code of the DASIF data base (extract)

APPENDIX XI

```
'*******************************************************'
'                GLOBAL VARS                '
'*******************************************************'

' Declare global container vars
Dim repDict As Scripting.Dictionary
Dim compDict As Scripting.Dictionary
Public LogList As Collection

' Declare gloal XML vars
Dim BOMxlpath As String
Dim MyBOMXL As Object
Dim NoDMDCounter As Integer
Dim answer As Integer
Dim xmldoc As DOMDocument
Dim xmlModels As IXMLDOMElement
Dim xmlComps As IXMLDOMElement


'*******************************************************'
'                MAIN ROUTINE                '
'*******************************************************'

Sub CATMain()
' Entry point of export macro.
'
' HOW TO :
' ========
' * Start associated form

   ' Init and show the main form


   ExportXML.Init_form

End Sub


'*******************************************************'
'                CORE ROUTINES                '
'*******************************************************'

Public Sub Export(ByVal filepath As String)
' Sub that export ActiveDocument's product description to an XML file.
'
' Parameters :
'   - FilePath : The path of the file to export.

   ' Initialize Global vars
   Set repDict = New Scripting.Dictionary
   Set compDict = New Scripting.Dictionary
   Set LogList = New Collection
   Set xmldoc = New DOMDocument

   ' Log Process Start
   Call Misc.LogInfo(LogList, "Début du processus d'export XML.")


   ' Get the active Product    ' Create XML structure
   Misc.LogInfo LogList, "Construction de la structure du fichier XML..."
```

```
Dim xmlRoot As IXMLDOMElement
Set xmlRoot = xmldoc.createElement(XML_TAGS.TAG_XMLROOT)
xmldoc.appendChild xmlRoot
Set xmlComps = xmldoc.createElement(XML_TAGS.TAG_COMPS)
xmlRoot.appendChild xmlComps
Set xmlModels = xmldoc.createElement(XML_TAGS.TAG_MODELS)
xmlRoot.appendChild xmlModels
Dim xmlLinks As IXMLDOMElement
Set xmlLinks = xmldoc.createElement(XML_TAGS.TAG_LINKS)
xmlRoot.appendChild xmlLinks


Misc.LogInfo LogList, "Identification du produit à traiter..."
On Error Resume Next
Dim prd As Product
Set prd = CATIA.ActiveDocument.Product
If Err.Number <> 0 Then
   ' Abort Process
   Misc.LogError LogList, "Impossible d'identifier le produit à traiter. Fin du processus."
   Exit Sub
Else
   ' Reset Error Handler
   On Error GoTo 0
End If

' Compute operation count
ExportXML.ComputeTickCount prd

NoDMDCounter = 0
' Generate Active Product XML description

GenXMLComponent prd, xmlComps

GenXMLLinks xmlLinks

' Generate the XML file
Dim retry As Boolean
Misc.LogInfo LogList, "Génération du fichier xml..."
ExportXML.status "Generating XML file..."
On Error Resume Next

Do
   ' save the file
   xmldoc.Save filepath
   ' If an error occured
   If Err.Number <> 0 Then
      ' Log the error
      Misc.LogWarning LogList, "Impossible d'écrire sur le fichier de sortie spécifié."
      ' Ask if user want to retry

      Dim choice As Integer
      choice = MsgBox("Impossible d'écrire sur le fichier de sortie spécifié.", vbCritical Or vbRetryCancel, "Erreur")
      ' If user asks retry
      If choice = 4 Then
         filepath = CATIA.FileSelectionBox("Save export file...", "*.xml", CatFileSelectionModeSave)
         xmldoc.Save filepath
         ' Log and continue the Loop
         Misc.LogInfo LogList, "Nouvel essai de génération du fichier xml..."
         retry = True
      Else
         ' Log and stop the loop
         Misc.LogError LogList, "Abandon de la génération du fichier XML."
         retry = False
      End If
```

```
      ' Else : no error
      Else
         ' Log and stop the loop
         Misc.LogInfo LogList, "Génération du fichier xml terminée."
         retry = False
      End If
   Loop While retry = True

   'Reset Error Handler
   On Error GoTo 0

   Dim ratio As Double
   ratio = NoDMDCounter * 100 / repDict.count
   MsgBox (trunc(ratio) & "% des articles de la DMU exportée n'ont aucun DMD spécifié")

   Dim YN As Integer
   Dim fpath As String
   Dim xlapp As Object
   Dim xlNoDMD As Object
   YN = MsgBox("Voulez-vous les lister dans un fichier Excel?", vbYesNo, "Lister les articles sans DMD?")
   If YN = vbYes Then
      Set xlapp = GetObject(, "Excel.Application")
      xlapp.Visible = True
      xlapp.UserControl = True
      Call xlapp.Workbooks.Add
      Dim XlSheet As Object
      Set XlSheet = xlapp.ActiveWorkbook.ActiveSheet
      XlSheet.Cells(1, 1).Value = "Item/Model_Ref"
      XlSheet.Cells(1, 2).Value = "Model_File_Path"

      Dim Model_List As IXMLDOMNode
      Dim node As IXMLDOMNode
      Dim dmdattr As IXMLDOMAttribute
      Dim nameattr As IXMLDOMAttribute
      Dim fileattr As IXMLDOMAttribute
      Dim RowIndex As Integer
      RowIndex = 1
      Set Model_List = xmldoc.selectSingleNode("//XML_ENRICHED_DMU_EXPORT/MODEL_LIST/").selectSingleNode(".//MODEL_LIST")
      For Each node In Model_List.childNodes
         If node.Attributes.Length <> 3 Then
            Set dmdattr = node.Attributes.getNamedItem("DMD_BOM_IfNotDefined")
            If dmdattr.nodeValue = "Unknown_DMD" Then
               Set nameattr = node.Attributes.getNamedItem("Name")
               Set fileattr = node.Attributes.getNamedItem("File")
               RowIndex = RowIndex + 1
               XlSheet.Cells(RowIndex, 1).Value = nameattr.nodeValue
               XlSheet.Cells(RowIndex, 2).Value = fileattr.nodeValue
            End If
         End If
      Next
   End If


   ' Clean up dicts
   Set repDict = Nothing
   Set compDict = Nothing

End Sub

Sub GenXMLComponent(ByRef component, ByRef xmlParent As IXMLDOMElement)
' Create an xml description of a Product.
'
' HOW TO :
' ========
```

```
' * Start sub that recursivly create XML component elements from the root component.
'
' Parameters :
' ============
'   - component : The product to describe.
'   - xmlParent : The XML parent node of the description.

    ' Log Start operation
    ExportXML.status "Saving product's components..."
    Misc.LogInfo LogList, "Traitements des information du produit en cours..."

    ' Start Recursion generation
    RecGenXMLComponent component, xmlParent

    ' Log End operation
    Misc.LogInfo LogList, "Traitements des information du produit terminé : " _
        & compDict.count & " Composants et " & repDict.count & " representations."

End Sub

Sub RecGenXMLComponent(ByRef component, ByRef xmlParent As IXMLDOMElement)
' Create an xml description of a Product. (inderect recursion)
'
' HOW TO :
' ========
' * If the component isn't registered yet :
'       * Register it.
'       * Create a new xml Component element and add it to the input xmlParent element.
'       * Call sub that Create an XML description of the component representation.
'       * Call sub that create an XML description of the component position.
'       * Call sub that create an XML description of the children components.
'
' Parameters :
' ============
'   - component : The product to describe.
'   - xmlParent : The XML parent node of the description.

    ' Declare XML objects
    Dim xmlComp As IXMLDOMElement
    Dim xmlAttr As IXMLDOMAttribute

    ' Declare and initialize ID var
    Dim compId As Long
    compId = 0

    ' Register the component if it is not already registered
    If Not compDict.Exists(component) Then
        ' Register it
        compId = compDict.count + 1
        compDict.Add component, compId


        ' Create XML objects
        Set xmlComp = xmldoc.createElement(XML_TAGS.TAG_COMP)
        xmlParent.appendChild xmlComp
        ' ID Attribute
        Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_ID)
        xmlAttr.nodeValue = compId
        xmlComp.setAttributeNode xmlAttr

        'Generate XML model attribute
        GenXmlModel component, xmlComp

        ' Instance_ID Attribute
```

```
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_ID)
    xmlAttr.nodeValue = component.name
    xmlComp.setAttributeNode xmlAttr
    ' ID Attribute
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_NAME)
    xmlAttr.nodeValue = component.DescriptionRef
    xmlComp.setAttributeNode xmlAttr
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_SNS)
    xmlAttr.nodeValue = component.Nomenclature
    xmlComp.setAttributeNode xmlAttr
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_REF)
    xmlAttr.nodeValue = component.PartNumber
    xmlComp.setAttributeNode xmlAttr
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_VERSION)
    xmlAttr.nodeValue = component.Revision
    xmlComp.setAttributeNode xmlAttr
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_FULLNAME)
    xmlComp.setAttributeNode xmlAttr
    xmlAttr.nodeValue = component.PartNumber & "_" & component.DescriptionRef & "_" & component.Definition & "_" & component.Re-
vision & "_" & component.Nomenclature


    ' Generate XML position node
    GenXmlComponentPosition component, xmlComp

    ' Generate XML Mass_Properties node
    GenXmlComponentMassProp component, xmlComp

    ' Generate XML children node
    GenXmlComponentChildren component, xmlComp

  End If

End Sub

Sub GenXmlModel(ByRef component, ByRef xmlParent As IXMLDOMElement)
' Sub that create an XML description of a product representation.
'
' HOW TO :
' ========
' * If the component representation isn't registered yet :
'     * Register it.
'     * Create a new xml Representation element and add it to the input xmlParent element.
' * Else :
'     * Get the ID of the registered representation.
' * Add the ID of the representation to the input xmlParent Element.
'
' Parameters :
' ============
'   - component : The product to describe.

  ' Declare XML objects
  Dim xmlModel As IXMLDOMElement
  Dim xmlBody As IXMLDOMElement
  Dim xmlAttr As IXMLDOMAttribute
  Dim fso As New FileSystemObject


  ' Declare and initialize ID var
  Dim name As String
  Dim repID
  Dim rep

  ' If the component has a representation
```

```
  If component.HasAMasterShapeRepresentation() Then
    ' Get the representation
    Set rep = component.GetShapeRepresentation(True, component.GetActiveShapeName(), catRep3D, True)
    ' If not already registered
    If Not repDict.Exists(rep) Then
      ' Register it
      repID = repDict.count + 1
      repDict.Add rep, repID
      ' Switch on document type
      Select Case TypeName(rep)
        ' If Part representation
        Case "PartDocument"
          name = rep.Part.name
        ' If other representation
        Case "Document"
          'Switch on file type
          Select Case UCase(fso.GetExtensionName(rep.FullName))
            ' CGR file
            Case "CGR"
              name = fso.GetBaseName(rep.FullName)
            ' STL file
            Case "STL"
              name = "Assembly"
          End Select
      End Select

      ' Generate Xml representation element
      Set xmlModel = xmldoc.createElement(XML_TAGS.TAG_MODEL)
      xmlModels.appendChild xmlModel
      ' ID Attribute
      Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_MODEL_ID)
      xmlAttr.nodeValue = repID
      xmlModel.setAttributeNode xmlAttr
      ' Name Attribute
      Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_MODEL_NAME)
      xmlAttr.nodeValue = name
      xmlModel.setAttributeNode xmlAttr
      ' File Attribute
      Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_CAD_FILE)
      xmlAttr.nodeValue = rep.FullName
      xmlModel.setAttributeNode xmlAttr

      'Create Bodies xml elements under Model elements
      GenXmlModelBodies rep, xmlModel

    ' Else get the representation's ID
    Else
      repID = repDict.Item(rep)
    End If

  ' Else set representation ID to 0
  Else
    repID = "No representation"

  End If

  ' RepID attribute of the parent node
  Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_MODEL)
  xmlAttr.nodeValue = repID
  xmlParent.setAttributeNode xmlAttr

End Sub

Sub GenXmlModelBodies(ByRef model, ByRef xmlParent As IXMLDOMElement)
```

```
Dim xmlBodies As IXMLDOMElement
Dim xmlBody As IXMLDOMElement
Dim xmlAttr As IXMLDOMAttribute
Dim xlapp As Object
Dim prd As Product
Set prd = CATIA.ActiveDocument.Product

Dim prt As Part
Set prt = model.Part

' Generate XML children element
Set xmlBodies = xmldoc.createElement(XML_TAGS.TAG_BODIES)
xmlParent.appendChild xmlBodies
' Gen count attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODIES_COUNT)
xmlAttr.nodeValue = prt.Bodies.count()
xmlBodies.setAttributeNode xmlAttr

If prt.Bodies.count <> 0 Then
   ' For each body defined in the model of the component, capture in the XML the mass, volume and material properties of the body
   Dim body As body
   Dim objSPAWkb As AnyObject
   Set objSPAWkb = CATIA.ActiveDocument.GetWorkbench("SPAWorkbench")
   Dim objRef As Reference
   Dim objMeasurable As Measurable
   Dim MyInertias As Inertias
   Dim myInertia As Inertia
   Dim M, V, MassVol As Double
   Dim dmddensity As String
   Dim dmd As Material
   Dim oManager As MaterialManager
   Set oManager = prd.GetItem("CATMatManagerVBExt")
   Dim counter As Integer
   counter = 0
   For Each body In prt.Bodies
      Set xmlBody = xmldoc.createElement(XML_TAGS.TAG_BODY)
      xmlBodies.appendChild xmlBody
      Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_NAME)
      xmlAttr.nodeValue = body.name
      xmlBody.setAttributeNode xmlAttr
      If body.name = prt.MainBody.name Then
         Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_MAIN)
         xmlAttr.nodeValue = "True"
         xmlBody.setAttributeNode xmlAttr
      Else
         Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_MAIN)
         xmlAttr.nodeValue = "False"
         xmlBody.setAttributeNode xmlAttr
      End If
      Set objRef = prt.CreateReferenceFromObject(body)
      Set objMeasurable = objSPAWkb.GetMeasurable(objRef)
      On Error Resume Next
         V = objMeasurable.Volume
      If Err.Number <> 0 Then
         V = 0
      End If
      Err.Clear
      On Error GoTo 0

      Set MyInertias = objSPAWkb.Inertias
      Set myInertia = MyInertias.Add(body)
      On Error Resume Next
      M = myInertia.Mass
```

```
    If Err.Number <> 0 Then
      M = Nothing
    End If
    Err.Clear
    On Error GoTo 0

    MassVol = myInertia.Density
    oManager.GetMaterialOnBody body, dmd

    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_MASS)
    xmlAttr.nodeValue = M
    xmlBody.setAttributeNode xmlAttr

    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_VOL)
    xmlAttr.nodeValue = V
    xmlBody.setAttributeNode xmlAttr

    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_DMD)

    On Error Resume Next
    xmlAttr.nodeValue = dmd.name
    If Err.Number <> 0 Then
      xmlAttr.nodeValue = "Unknown_DMD"
      xmlBody.setAttributeNode xmlAttr
      Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_DENSITY)
      xmlAttr.nodeValue = 8000
      xmlBody.setAttributeNode xmlAttr
      counter = counter + 1
    Err.Clear
    On Error GoTo 0
    Else
      xmlBody.setAttributeNode xmlAttr

      Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_BODY_DENSITY)
      xmlAttr.nodeValue = MassVol
      xmlBody.setAttributeNode xmlAttr
    End If
  Next

  'procédure nécessaire lorsque les DMD ne sont pas renseignés : récupérer d'une nomenclature un DMD et/ou une masse indicative à mettre comme propriété de l'article
  If prt.Bodies.count = counter Then
    Dim dmd2
    Dim DMDFound As Boolean

    If BOMxlpath = "" Then
      Call MsgBox("La DMU exportée contient des modèles CAO sans DMD et aucune nomenclature n'a été fournie!", vbExclamation)
      answer = MsgBox("Souhaitez-vous importer les DMD depuis une nomenclature?", vbYesNo, "Do you need a BOM to import DMD parameters?")
      If answer = vbYes Then
        Set xlapp = CreateObject("Excel.Application")
        BOMxlpath = CATIA.FileSelectionBox("Veuillez-sélectionner le fichier Excel de nomenclature contenant les DMD...", "*.xlsx", CatFileSelectionModeOpen)
        Set MyBOMXL = GetObject(BOMxlpath)
        MyBOMXL.Application.Visible = True
        MyBOMXL.Parent.Windows(1).Visible = True
      End If
    End If

    'Call MsgBox("Aucun DMD n'est défini pour l'article : " & prt.name, vbExclamation, "No DMD found")

    'lancement d'une procédure permettant de recupérer les DMD d'une nomenclature et de renvoyer la vaelur du DMD appliquée à la pièce
    If answer = vbYes Then
```

```
        Call applyDMD.GetDMDtoApply(MyBOMXL, prt, dmd2, dmddensity, DMDFound)

        Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_MODEL_DMD)

        If DMDFound = False Then
          xmlAttr.nodeValue = "Unknown_DMD"
          xmlParent.setAttributeNode xmlAttr
          Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_DMD_DENSITY)
          xmlAttr.nodeValue = 8000
          xmlParent.setAttributeNode xmlAttr
          NoDMDCounter = NoDMDCounter + 1
        Else

          xmlAttr.nodeValue = dmd2.name
          xmlParent.setAttributeNode xmlAttr

          Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_DMD_DENSITY)
          xmlAttr.nodeValue = MassVol
          xmlParent.setAttributeNode xmlAttr

          'capture de la référence DMD au niveau du modèle
          Call applyDMD.applyDMD(prd, prt, dmd2)

          'appliquer la densité correspondante aux corps de pièces enfants
          For Each body In prt.Bodies
            'MsgBox (Body.name)
            Set myInertia = MyInertias.Add(body)
            MassVol = myInertia.Density
            Dim node As IXMLDOMElement
            Dim xmldensity As IXMLDOMAttribute
            For Each node In xmlBodies.childNodes
              Set xmldensity = node.getAttributeNode(XML_TAGS.ATT_BODY_DENSITY)
              'MsgBox (node.Attributes.item(0).nodeValue)
              xmldensity.nodeValue = MassVol
            Next
          Next

        End If

      End If

    End If

End If

End Sub



Sub GenXmlComponentPosition(ByRef component, ByRef xmlParent As IXMLDOMElement)
' Sub that create an xml description of a product's position.
'
' HOW TO :
' ========
' * Get the component axis system.
' * Create a new xml Position element and add it to the input xmlParent element.
' * Create xml elements for Origin, axis X, axis y and axis Z and ad them to the position element.
'
' Parameters :
' ============
'   - component : The product to describe.
'   - xmlParent : The XML parent node of the description.

    ' Declare XML objects
```

```
Dim xmlPos As IXMLDOMElement
Dim xmlTVector, xmlRMatrix, xmlVectX, xmlVectY, xmlVectZ As IXMLDOMElement
Dim xmlAttr As IXMLDOMAttribute
Dim pos(11)

' Register the Component position
component.Position.GetComponents pos
' Generate XML Position Element
Set xmlPos = xmldoc.createElement(XML_TAGS.TAG_COMP_POS)
xmlParent.appendChild xmlPos

' Generate Translation Vector of the local axis system of the component
Set xmlTVector = xmldoc.createElement(XML_TAGS.TAG_COMP_POS_T)
xmlPos.appendChild xmlTVector
' Gen X component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_POS_TX)
xmlAttr.nodeValue = pos(9)
xmlTVector.setAttributeNode xmlAttr
' Gen Y component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_POS_TY)
xmlAttr.nodeValue = pos(10)
xmlTVector.setAttributeNode xmlAttr
' Gen Z component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_POS_TZ)
xmlAttr.nodeValue = pos(11)
xmlTVector.setAttributeNode xmlAttr

' Generate Rotation Matrix
Set xmlRMatrix = xmldoc.createElement(XML_TAGS.TAG_COMP_POS_R)
xmlPos.appendChild xmlRMatrix
' Generate Vector X of Rotation Matrix
Set xmlVectX = xmldoc.createElement(XML_TAGS.TAG_COMP_POS_RX)
xmlRMatrix.appendChild xmlVectX
' Gen X component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_1)
xmlAttr.nodeValue = pos(0)
xmlVectX.setAttributeNode xmlAttr
' Gen Y component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_2)
xmlAttr.nodeValue = pos(1)
xmlVectX.setAttributeNode xmlAttr
' Gen Z component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_3)
xmlAttr.nodeValue = pos(2)
xmlVectX.setAttributeNode xmlAttr


' Generate Vector Y of Rotation Matrix
Set xmlVectY = xmldoc.createElement(XML_TAGS.TAG_COMP_POS_RY)
xmlRMatrix.appendChild xmlVectY
' Gen X component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_1)
xmlAttr.nodeValue = pos(3)
xmlVectY.setAttributeNode xmlAttr
' Gen Y component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_2)
xmlAttr.nodeValue = pos(4)
xmlVectY.setAttributeNode xmlAttr
' Gen Z component Attribute
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_3)
xmlAttr.nodeValue = pos(5)
xmlVectY.setAttributeNode xmlAttr

' Generate Vector Z of Rotation Matrix
```

```
    Set xmlVectZ = xmldoc.createElement(XML_TAGS.TAG_COMP_POS_RZ)
    xmlRMatrix.appendChild xmlVectZ
    ' Gen X component Attribute
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_1)
    xmlAttr.nodeValue = pos(6)
    xmlVectZ.setAttributeNode xmlAttr
    ' Gen Y component Attribute
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_2)
    xmlAttr.nodeValue = pos(7)
    xmlVectZ.setAttributeNode xmlAttr
    ' Gen Z component Attribute
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_TAG_POS_3)
    xmlAttr.nodeValue = pos(8)
    xmlVectZ.setAttributeNode xmlAttr

End Sub

Sub GenXmlComponentChildren(ByRef component, ByRef xmlParent As IXMLDOMElement)
' Sub that create an xml description of product's children.
'
' HOW TO :
' ========
' * If the Component has Children.
' * Create a new xml Children list element and add it to the input xmlParent element.
' * For each component in the children list :
'       * Call recursive sub that create XML component description.
'
' Parameters :
' ============
'   - component : The product to describe.
'   - xmlParent : The XML parent node of the children description.

    ' Declare XML objects
    Dim xmlChildren As IXMLDOMElement
    Dim xmlAttr As IXMLDOMAttribute


    ' Generate XML children element
    Set xmlChildren = xmldoc.createElement(XML_TAGS.TAG_COMP_CHILDREN)
    xmlParent.appendChild xmlChildren
    ' Gen count attribute
    Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_CHILDREN_COUNT)
    xmlAttr.nodeValue = component.Products.count
    xmlChildren.setAttributeNode xmlAttr

    ' If the component has children
    If component.Products.count <> 0 Then

        ' For each subproduct generate a new component description
        Dim child As Object
        For Each child In component.Products
            RecGenXMLComponent child, xmlChildren
        Next child

    Else
        ' Tick the status form
        ExportXML.Tick

    End If

End Sub

Sub GenXmlComponentMassProp(ByRef component, ByRef xmlParent As IXMLDOMElement)
```

```
' Declare XML objects
Dim xmlMassProp As IXMLDOMElement
Dim xmlMass As IXMLDOMElement
Dim xmlCG As IXMLDOMElement
Dim xmlInertia, xmlInertiaX, xmlInertiaY, xmlInertiaZ As IXMLDOMElement
Dim xmlAttr As IXMLDOMAttribute

Dim myInertia
Set myInertia = component.GetTechnologicalObject("Inertia")
Dim coordCG(2), matrixInertia(8)
myInertia.GetCOGPosition coordCG
myInertia.GetInertiaMatrix matrixInertia

' Generate XML Mass properties elements
Set xmlMassProp = xmldoc.createElement(XML_TAGS.TAG_COMP_MASS_PROP)
xmlParent.appendChild xmlMassProp

Set xmlMass = xmldoc.createElement(XML_TAGS.TAG_COMP_MASS)
xmlMassProp.appendChild xmlMass
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_MASS_VALUE)
xmlAttr.nodeValue = myInertia.Mass
xmlMass.setAttributeNode xmlAttr
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_MASS_UNIT)
xmlAttr.nodeValue = "kg"
xmlMass.setAttributeNode xmlAttr

Set xmlCG = xmldoc.createElement(XML_TAGS.TAG_COMP_CG)
xmlMassProp.appendChild xmlCG
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_CG_X)
xmlAttr.nodeValue = coordCG(0)
xmlCG.setAttributeNode xmlAttr
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_CG_Y)
xmlAttr.nodeValue = coordCG(1)
xmlCG.setAttributeNode xmlAttr
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_CG_Z)
xmlAttr.nodeValue = coordCG(2)
xmlCG.setAttributeNode xmlAttr
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_CG_UNIT)
xmlAttr.nodeValue = "m"
xmlCG.setAttributeNode xmlAttr

Set xmlInertia = xmldoc.createElement(XML_TAGS.TAG_COMP_INERTIA)
xmlMassProp.appendChild xmlInertia
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_UNIT)
xmlAttr.nodeValue = "kg.m²"
xmlInertia.setAttributeNode xmlAttr

Set xmlInertiaX = xmldoc.createElement(XML_TAGS.TAG_COMP_INERTIAX)
xmlInertia.appendChild xmlInertiaX
Set xmlInertiaY = xmldoc.createElement(XML_TAGS.TAG_COMP_INERTIAY)
xmlInertia.appendChild xmlInertiaY
Set xmlInertiaZ = xmldoc.createElement(XML_TAGS.TAG_COMP_INERTIAZ)
xmlInertia.appendChild xmlInertiaZ

Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IXX)
xmlAttr.nodeValue = matrixInertia(0)
xmlInertiaX.setAttributeNode xmlAttr
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IXY)
xmlAttr.nodeValue = matrixInertia(1)
xmlInertiaX.setAttributeNode xmlAttr
Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IXZ)
xmlAttr.nodeValue = matrixInertia(2)
xmlInertiaX.setAttributeNode xmlAttr
```

```
   Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IYX)
   xmlAttr.nodeValue = matrixInertia(3)
   xmlInertiaY.setAttributeNode xmlAttr
   Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IYY)
   xmlAttr.nodeValue = matrixInertia(4)
   xmlInertiaY.setAttributeNode xmlAttr
   Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IYZ)
   xmlAttr.nodeValue = matrixInertia(5)
   xmlInertiaY.setAttributeNode xmlAttr

   Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IZX)
   xmlAttr.nodeValue = matrixInertia(6)
   xmlInertiaZ.setAttributeNode xmlAttr
   Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IZY)
   xmlAttr.nodeValue = matrixInertia(7)
   xmlInertiaZ.setAttributeNode xmlAttr
   Set xmlAttr = xmldoc.CreateAttribute(XML_TAGS.ATT_COMP_INERTIA_IZZ)
   xmlAttr.nodeValue = matrixInertia(8)
   xmlInertiaZ.setAttributeNode xmlAttr


End Sub
...
```

Table 20: CATIA macro for generating the DMU exported XML file

APPENDIX XII

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 12 13:21:33 2013

@author: s064161
"""
from __future__ import print_function

# Import modules
import os.path
import xml.etree.ElementTree as etree
import datetime
import datamodel
import math
import numpy as np


#Shortcuts creation for rapid access to database entities references
global DesignDef
DesignDef=datamodel.SystemDefinition.DesignDefinition
global nauo
nauo=datamodel.SystemDefinition.NAUO
global Item Part
Item_Part=datamodel.SystemDefinition.ItemPart
global Item_Assy
Item_Assy=datamodel.SystemDefinition.ItemAssembly
global Item StrAssy
Item StrAssy=datamodel.SystemDefinition.StructuralAssembly
global root
root=datamodel.SystemDefinition.Root
global CADModel
CADModel=datamodel.SystemDefinition.CADModel
...
global ClassMatingFeature
ClassMatingFeature=datamodel.SystemDefinition.MatingFeaturePublication
global ClassShapeContact
ClassShapeContact=datamodel.SystemDefinition.ShapeContact
global ClassInterference
ClassInterference=datamodel.SystemDefinition.FunctionalInterference
global ClassClearance
ClassClearance=datamodel.SystemDefinition.FunctionalClearance
global ClassDof
ClassDof=datamodel.SystemDefinition.Dof


# Connection to the existing database
#datamodel.db.connect("localhost","db_test_tv","tvosgien","cool01")
#Or creation of a new data base
datamodel.db.new("localhost","db_test_tv","tvosgien","cool01")

#Create Organisations instances
orga = datamodel.ConfManagement.organisation.Organisation
orga_entityM=orga(name="Snecma_WRS")
orga entityM.integrator= True
orga entityYY=orga(name="YY")
orga entityYY.mother=orga entityM
orga_entityYYT=orga(name="YYT")
orga_entityYYT.mother=orga_entityYY
orga entityYYTD=orga(name="YYTD")
orga entityYYTD.mother=orga entityYYT


#Create a product, a configuration and view definition contexts instances
prd=datamodel.ConfManagement.conf.ProductClass

prd entitySC=prd(name="SilverCrest", prdtype="Business Jet Engines")
prd entitySC.orgas.append(orga entityM)

confgen=datamodel.ConfManagement.conf.ManufacturableConfiguration
confdes=datamodel.ConfManagement.conf.DesignDisciplineConfiguration

conf entityCT2=confgen(name="CT2")
conf_entityCT2.description="SC_CORE_TEST_2"
conf_entityCT2.creation_date = datetime.datetime.today()
```

```
conf_entityYYTD=confdes(name="Mechanical Integration YYTD")
conf_entityYYTD.description="SC_CT_Dynamique Ensemble_YYTD"
conf_entityYYTD.base_conf=conf_entityCT2
conf entityYYTD.creation date = datetime.datetime.today()

prd entitySC.confs.append(conf entityCT2)
prd_entitySC.confs.append(conf_entityYYTD)

context=datamodel.__local__.ViewDefinitionContext

view context1=context(life cycle stage= "Detailed Design", application domain="Physical Test")
view context1.description="Conf Core-Test Generique"
view_context1.confs.append(conf_entityCT2)
view_context2=context(life_cycle_stage= "Detailed Design", application_domain="Mechanical De-
sign YYTD")
view context2.description="Vue dynamique d'ensemble YYTD"
view context2.confs.append(conf entityYYTD)


conf_list=[]
conf list.append(conf entityCT2)

global EffList_entity
EffList_entity=ClassEffList(conf_list)


#Commit the transaction of new created objects to the database
datamodel.db.session.commit()


#Child-Parent components dictionary
global parent map
parent map=dict()

global doc

global nauo list
nauo list=[]

# Fill the data base with data extracted from a DMU export xml file
def imp_dmu_from_xml(file_path):
    """Import the DMU definition from a DMU xml export file and create the
    corresponding items, components, models, linkages in the database"""

    # Compute file's absolute path
    abs_path = os.path.abspath(file_path)
    print("Import : '{}'".format(abs_path))
    doc = etree.parse(abs path)

    #Find the node 'Component List'
    comp_list_elt=doc.find(".//COMPONENT_LIST")
    #Get the root component
    comp_root=comp_list_elt.find("COMPONENT")

    #Find the node 'Model list'
    model list=doc.find(".//MODEL LIST")

    #Find the node 'Link_list'
    link_list=doc.find(".//LINKAGE_LIST")

    #Call parse comp function with the root component as argument
    parse_comp(comp_root,None, model_list)

    #Update the data base with the created instances
    datamodel.db.session.commit()

    #Call parse_links function to retrieve information about components' linkages
    parse_links(model_list, link_list)

    #Update the data base with the created instances
    datamodel.db.session.commit()



def parse_comp(comp, nauo_parent, models):
    '''
    Recursive function to collect all components definition features present in the xml file
including hierarchy, definition, models, mass properties, interfaces, etc.
    '''
    #get component attributes
```

```
    comp_attr=comp.attrib
    comp_name=comp_attr['InstanceID']
    sns=comp_attr['SNS']
    description= comp attr['Description']
    configurable = True
    ref item = comp attr['Ref Article']
    index = "A"
    nversion = comp_attr['Indice']
    status="Frozen"
    FullName= comp attr['FullName']

    #Find the 'Children List' node in xml
    comp_children_elt=comp.find(".//CHILDREN_LIST")
    children_count= len(list(comp_children_elt))

    #Create DesignDefinition corresponding entity required to construct the NAUO instance
        #check if the artifact and related definition already exist in the database
    if datamodel.SystemDefinition.NAUO.get_by(fullname=FullName) == None:
        def_exist=False
        DesignDef_entity=DesignDef(nversion, status, view_context1)

        #Create the NAUO instance
        nauo_entity=nauo(comp_name, sns, DesignDef_entity)
        nauo_entity.description=description
        nauo entity.fullname=FullName
        nauo entity.configurable=configurable
        nauo entity.effs.append(EffList entity)


        #Creation of the corresponding DesignArtifact entities
        if comp name != "ROOT-SC":
            if children count == 0:
                item part entity=Item Part(description, ref item, index)
                item part entity.defs.append(DesignDef entity)
            else:
                item_assy_entity=Item_Assy(description, ref_item, index)
                item assy entity.defs.append(DesignDef entity)
        else:
            root entity=root(description, conf entityCT2)
            root_entity.defs.append(DesignDef_entity)

        #Create the Nauo's parent instance
        if nauo parent is not None:
            nauo parent.definition.children.append(nauo entity)

    else:
        #The definition already exist so a new NAUO of this definition is instantiated
        previous nauo=datamodel.SystemDefinition.NAUO.get by(fullname=FullName)
        DesignDef entity=previous nauo.definition
        nauo entity=nauo(comp name, sns, DesignDef entity)
        nauo_entity.description=description
        nauo_entity.fullname=FullName
        nauo_entity.configurable=configurable
        nauo entity.effs.append(EffList entity)

        nauo parent.definition.children.append(nauo entity)
        def_exist=True

    #Retrieve instance's position and properties (Center of mass and Inertia Moments)
        #retrieve position
    position node=comp.find('.//POSITION')
    node_trans=position_node.find('.//Translation')
    position_entity=ClassPosition(nauo_entity)
    position_entity.unit="Degrees"

    position entity.trans x=float(node trans.attrib['TX'])
    position_entity.trans_y=float(node_trans.attrib['TY'])
    position_entity.trans_z=float(node_trans.attrib['TZ'])
    node_rx=position_node.find('.//Rotation/RX')
    node_ry=position_node.find('.//Rotation/RY')
    node rz=position node.find('.//Rotation/RZ')
    position entity.rotxx=float(node rx.attrib['X'])
    position_entity.rotxy=float(node_rx.attrib['Y'])
    position_entity.rotxz=float(node_rx.attrib['Z'])
    position_entity.rotyx=float(node_ry.attrib['X'])
    position_entity.rotyy=float(node_ry.attrib['Y'])
    position entity.rotyz=float(node ry.attrib['Z'])
    position entity.rotzx=float(node rz.attrib['X'])
    position_entity.rotzy=float(node_rz.attrib['Y'])
    position_entity.rotzz=float(node_rz.attrib['Z'])
```

```
        #retrieve instance's mass properties
    mass_prop_node=comp.find('.//MASS_PROPERTIES')
    cg node=mass prop node.find('.//CG')
    inertia node=mass prop node.find('.//INERTIA MATRIX')
    cgrel entity=ClassCGRel(nauo entity)
    inertia_entity1=ClassInertiaRel(nauo_entity)
    cgrel_entity.cgx=float(cg_node.attrib['CGX'])
    cgrel_entity.cgy=float(cg_node.attrib['CGY'])
    cgrel entity.cgz=float(cg node.attrib['CGZ'])
    cgrel entity.unit='m'

    #creation of the inertia matrix
    vectorx=[float(inertia_node.find('.//VectorIx').attrib['Ixx']),
             float(inertia node.find('.//VectorIx').attrib['Ixy']),
             float(inertia node.find('.//VectorIx').attrib['Ixz'])]
    vectory=[float(inertia node.find('.//VectorIy').attrib['Iyx']),
             float(inertia_node.find('.//VectorIy').attrib['Iyy']),
             float(inertia_node.find('.//VectorIy').attrib['Iyz'])]
    vectorz=[float(inertia_node.find('.//VectorIz').attrib['Izx']),
             float(inertia node.find('.//VectorIz').attrib['Izy']),
             float(inertia node.find('.//VectorIz').attrib['Izz'])]
    InertiaMatrix=np.array([vectorx, vectory, vectorz])

    #calculation of Inertia Matrix's eigenvalues
    lambda =np.linalg.eigvals(InertiaMatrix)

    #diagonalisation of the inertia matrix to obtain the moments according to main axes
    inertia_entity1.polar_x=float(inertia_node.find('.//VectorIx').attrib['Ixx'])-lambda_[0]
    inertia_entity1.diametral_y=float(inertia_node.find('.//VectorIy').attrib['Iyy'])-
lambda [1]
    inertia entity1.diametral z=float(inertia node.find('.//VectorIz').attrib['Izz'])-
lambda [2]
    inertia entity1.unit='kg.m2'

    #idenitfy the model of the component
    model id=comp.attrib['Model ID']
    mass node=mass prop node.find('.//Mass')

    #retrieve the mass of the component calculated from the geometry and material properties
    nauo_mass_entity=ClassInstanceMass(mass_node.attrib['Value'], nauo_entity)
    nauo mass entity.unit=mass node.attrib['MassUnit']

    if model id != "No representation":
        model=models.find(".//MODEL[@Model_ID='%s']" % (model_id))

        if CADModel.get_by(name=model.attrib['Name']) == None :
            file entity=CADFile(model.attrib['File'])
            start=str(model.attrib['File']).find('.')+1
            file entity.format=str(model.attrib['File'])[start:]

            model_entity=CADModel(model.attrib['Name'], model.attrib['Revision'],
                                  file_entity)

            DesignDef entity.models.append(model entity)


            #retrieve shape depandant properties
            body_list=models.find(".//MODEL/BODY_LIST")
            #retrieve volume, material and mass parameters for each body composing the model
            for body in list(body list):
                body_entity=ClassShapeBody(model_entity, body.attrib['Name'],
                                           to_bool(body.attrib['Main']))
                model_entity.bodies.append(body_entity)

                volume=float(body.attrib['Volume m3'])
                dmd_=body.attrib['DMD']
                rho=float(body.attrib['Density_kg.m3'])
                body_mass=float(body.attrib['Mass_kg'])

                #retrieve body volume parameter
                volume body entity=ClassVolume(volume, 'm3', DesignDef entity)
                body_entity.body_volume=volume_body_entity
                mass_body_entity=ClassCADMass(body_mass, 'kg', DesignDef_entity)
                body_entity.body_mass=mass_body_entity
                volume_body_entity.shape=model_entity
                mass body entity.shape=model entity

                #retrieve dmd parameter
                if ClassMaterial.get_by(refdmd=dmd_)==None:
```

```
                    dmd_entity=ClassMaterial(dmd_, DesignDef_entity)
                    #material_entity=ClassMaterial(dmd_entity.dmd, DesignDef_entity)
                    body_entity.material=dmd_entity
                    ClassDensity(dmd entity, rho, 'kg.m3', DesignDef entity)
                else:
                    body entity.material=ClassMaterial.get by(refdmd=dmd )

    #local mass properties for assemblies
    cg_abs_entity=ClassCGAbs(nauo_entity, DesignDef_entity)
    cg abs entity.unit='m'
    inertia abs entity=ClassInertiaAbs(nauo entity, DesignDef entity)
    inertia abs entity.unit='kg.m2'

    #assign nauo's effectivities
    nauo entity.effs.append(EffList entity)

    if not def exist:
        #apply the same procedure on children components
        for child in list(comp_children_elt):
            #fill the child-parent dictionnary
            parent map[child]=comp
            #Recursion on the children components
            parse_comp(child, nauo_entity, models)


    return parent map
    return nauo list


    '''
    Recursive function to collect all interfaces definition features present in the xml file
including interfaces attributes, interacting components, mating features and CAD interfaces
publications, etc.
   '''
def parse_links(model_list, link_list):
    for link in list(link_list):
        link name=link.attrib['Linkage Name']
        link direct=to bool(link.attrib['Direct'])
        link perma=to bool(link.attrib['Permanent'])
        link_domain=link.attrib['Domain']
        link_type=link.attrib['Linkage_Type']

        dof node=link.find('.//DOF')
        params node=link.find('.//LINKAGE PARAMETERS')

        topo_node=link.find('.//LINKAGE_TOPOLOGY')

      #Creation of associated Interaction instances
        if link domain == 'Mechanical':
            if link type=='Bolted Flange':
                nb_holes=params_node.attrib['Nb_holes']
                nb_comps=params_node.attrib['Nb_comps']
                alignment=to_bool(params_node.attrib['Centrage'])
                interaction entity=ClassBoltedFlange(nb comps, alignment,
                                                     nb holes)
            if link type=='Bolted Joint':
                interaction_entity=ClassBoltedJoint(type_bj='Bolted Joint')
            if link_type=='Bearing Joint':
                interaction_entity=ClassBearingJoint(direct=link_direct,
                                                     permanent=link perma,
                                                     can be dismantled=True,
                                                     techno='Ball bearing')
            if link_type=='Generic joint':
                generic_type=params_node.attrib['Interface_Type']
                interaction entity=ClassGenericJoint(direct=link direct,
                                                     permanent=link perma,
                                                     joint_type=generic_type)

        # Retrieve the associated set of degrees of freedom values
        dof_entity=ClassDof(interaction_entity)
        dof entity.Tx=to bool(dof node.attrib['Tx'])
        dof entity.Ty=to bool(dof node.attrib['Ty'])
        dof_entity.Tz=to_bool(dof_node.attrib['Tz'])
        dof_entity.Rx=to_bool(dof_node.attrib['Rx'])
        dof_entity.Ry=to_bool(dof_node.attrib['Ry'])
        dof_entity.Rz=to_bool(dof_node.attrib['Rz'])


        interface_entity=ClassInterface(link_name, interaction_entity)
```

```
        # creation of a dictionary of interacting components
        related_comps=dict()
        #Retrieve all defined and published mating features and associated publications
        for mating feature in list(topo node):
            topo type=mating feature.attrib['Topology Type']
            topo name=mating feature.attrib['Topology Name']
            prd1=mating_feature.find('.//FIRST_COMPONENT')
            prd1_name=prd1.attrib['Component_Name']
            prd1_id=prd1.attrib['Instance_Id']
            prd1 model=prd1.attrib['CAD Model Id']
            if prd1.attrib['Publication'] !="":
                pub1=prd1.attrib['Publication']

            prd2=mating_feature.find('.//SECOND_COMPONENT')
            prd2 name=prd2.attrib['Component Name']
            prd2 id=prd2.attrib['Instance Id']
            prd2 model=prd2.attrib['CAD Model Id']
            if prd1.attrib['Publication'] !="":
                pub2=prd2.attrib['Publication']


            nauo1=nauo.get by(name=prd1 id)
            if prd1_id not in related_comps:
                related_comps[prd1_id]=nauo1
            nauo2=nauo.get by(name=prd2 id)
            if prd2 id not in related comps:
                related comps[prd2 id]=nauo2

            mating_comps=[nauo1, nauo2]

            model1=model list.find(".//MODEL[@Model ID='%s']" % (prd1 model))
            model2=model list.find(".//MODEL[@Model ID='%s']" % (prd2 model))
            cao1=CADModel.get by(name=model1.attrib['Name'])
            pub1 entity=ClassMatingFeature(cao1, pub1)
            cao2=CADModel.get_by(name=model2.attrib['Name'])
            pub2_entity=ClassMatingFeature(cao2, pub2)

            #Treatment pre-defined mating features to build the interface templates
            if topo type=='contact':
                topo_entity=ClassShapeContact(topo_name, nauo1, nauo2)
                if link_type=='Bolted Flange':
                    if topo name[:16]=='Appui Plan Bride':
                        interface entity.mating features.append(topo entity)
                        interaction entity.bf planars.append(topo entity)
                    if topo_name[:14]=='Centrage_Bride':
                        interface_entity.mating_features.append(topo_entity)
                        interaction_entity.alignments.append(topo_entity)

                if link type=='Bearing Joint':
                    if topo name[:20]=='Contact Rotor Stator':
                        interface_entity.mating_features.append(topo_entity)
                        interaction_entity.contact_rotor_stators.append(topo_entity)
                    if topo_name[:19]=='Blocage_Translation':
                        interface entity.mating features.append(topo entity)
                        interaction entity.trans blockings.append(topo entity)

            if topo_type=='interference':
                topo_entity=ClassInterference(topo_name, nauo1, nauo2)
                interface_entity.mating_features.append(topo_entity)
            if topo type=='clearance':
                topo entity=ClassClearance(topo name, nauo1, nauo2)
                interface_entity.mating_features.append(topo_entity)

            topo_entity.mating_features.append(pub1_entity)
            topo entity.mating features.append(pub2 entity)

            interface_entity.mating_features.append(topo_entity)

        interface_entity.related_comps.extend(related_comps.values())

...
```

Table 21: Extract of the python script for enriching the database with test-case data set (from the DMU XML file generated from CATIA)

APPENDIX XIII

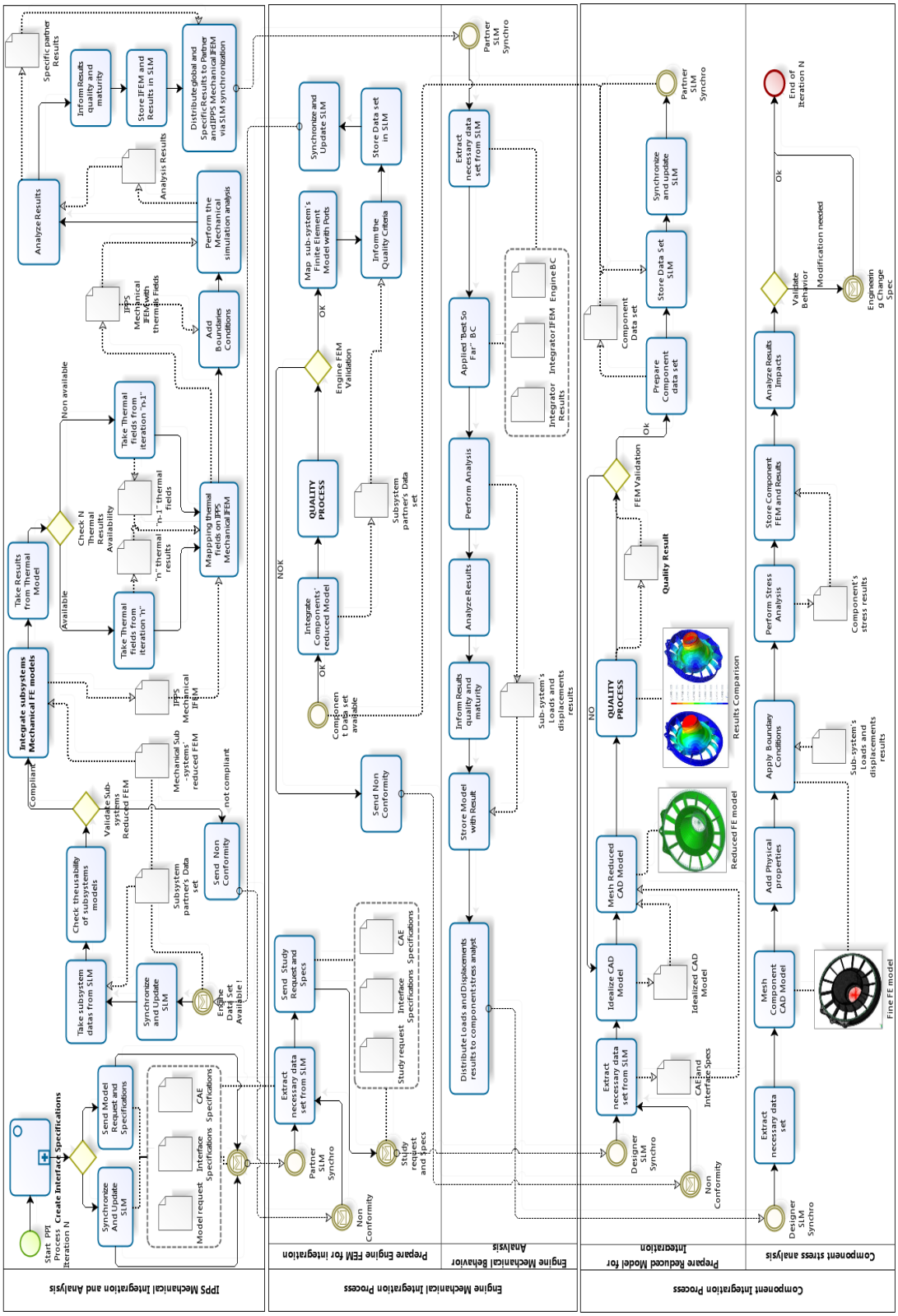


Figure 252: Product Integration process in BPMN

APPENDIX XIV

	PART A			INTERFACE					PART B		
	Part	**Mesh Element type**	**Publication/ port name**	**domain**	**type**	**techno**	**Ele-ment finite**	**Mesh el-ement type**	**Publication port name**	**Mesh element type**	**part**
IPPS system	Pylon	RBAR	P-N_FC1	mechanical	Bolted flange	Rivets	Node to node	RBAR	N-P_FC1	RBAR	nacelle
	Nacelle	RBAR	N-E_FCf1	mechanical	Bolted flange	rivet	Node to node	RBE2	E-N_FCf1	RBAR	Engine
	Nacelle	RBAR	N-E_FCR1	mechanical	Linear contact	v-groove	Node to node	RBE2	E-N_FCR1	RBAR	Engine
	Nacelle	RBAR	N-E_FCR2	mechanical	Planar contact	bolt	Node to node	RBE2	E-N_FCR2	RBAR	Engine
	Engine	RBE3 + CELAS1	E-P_TBHR	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_TBHR	CROD	Pylon
	Engine	RBE3 + CELAS1	E-P_TBHL	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_TBHL	CROD	Pylon
	Engine	RBE3 + CELAS1	E-P_TBHM	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_TBHM	CROD	Pylon
	Engine	RBE3 + CELAS1	E-P_ICR	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_HSR	CELAS+ RBAR	Pylon
	Engine	RBE3 + CELAS1	E-P_ICL	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_HSL	CELAS+ RBAR	Pylon
	Engine	RBE3 + CELAS1	E-P_FCL	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_FCL	CROD	Pylon
	Engine	RBE3 + CELAS1	E-P_FCR	mechanical	Bolted flange	Ball socket joint	Node to node	RBE2	P-E_FCR	CROD	Pylon
ENGINE system	Front Case	RBAR	FC-IC	mechanical	Bolted flange	Bolt	Node to node	CELAS	IC-FC	RBAR	InterCase
	Front Case	RBE2	FC-HPR	mechanical	Bearing	Ball bear-ing	Node to node	CELAS2-	HPR-FC	RBAR	HP Rotor

Table 22: Detailed interface specifications for PPS IFEM creation

APPENDIX XV

```
<?xml version="1.0" encoding="UTF-8" ?>
<fl:RFLPImportExport xsi:schemaLocation="RFLP.ImportExport RFLPImportExport.xsd" Role="VPLMDesigner" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" CATIAInfo="B212_3_0" Organisation="Company Name" xmlns:fl="RFLP.ImportExport" User="YFP" Project="Standard">
	<fl:ImplementLink Custo="CRE_Implement" Type="RefRef" ID="ID_Cnx_21" Name="Cnx_21" Modeler="RFLPLMImplementConnection">
		<fl:ObjectAttribute InternalName="E_export_id" Value="CRE_ImplCnx_21" Type="String" Name="E_Export_id" Mandatory="Y"/>
		<fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Cnx_21" Type="String" Name="Name" Mandatory="N"/>
		<fl:SourceCtx IDRef="ID_YFP_ROOT_Product"/>
		<fl:TargetCtx IDRef="ID_YFP_Root_Logical"/>
	</fl:ImplementLink>
	<fl:ImplementLink Custo="CRE_Implement" Type="PortPort" ID="ID_Cnx_Port Logical on Root-YFP_Publication_3D" Name="Cnx_Port Logical on Root-YFP_Publication_3D" Modeler="RFLPLMImplementConnection">
		<fl:ObjectAttribute InternalName="E_export_id" Value="CRE_ImplCnx_163" Type="String" Name="E_Export_id" Mandatory="Y"/>
		<fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Cnx_Port Logical on Root-YFP_Publication_3D" Type="String" Name="Name" Mandatory="N"/>
		<fl:SourceCtx IDRef="YFP_ROOT_Product"/>
		<fl:SourcePath>
			<fl:Object IDRef="Publication_00"/>
		</fl:SourcePath>
		<fl:TargetCtx IDRef="YFP_Root_Logical"/>
		<fl:TargetPath>
			<fl:Object IDRef="Port_Logical on RootL"/>
		</fl:TargetPath>
	</fl:ImplementLink>
	<fl:ImplementLink Custo="CRE_Implement" Type="PortPort" ID="ID_Cnx_sub-1" Name="Cnx_sub-1" Modeler="RFLPLMImplementConnection">
		<fl:ObjectAttribute InternalName="E_export_id" Value="CRE_ImplCnx_165" Type="String" Name="E_Export_id" Mandatory="Y"/>
		<fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Cnx_sub-1" Type="String" Name="Name" Mandatory="N"/>
		<fl:SourceCtx IDRef="YFP_ROOT_Product"/>
		<fl:SourcePath>
			<fl:Object IDRef=""/>
		</fl:SourcePath>
		<fl:TargetCtx IDRef="YFP_Root_Logical"/>
		<fl:TargetPath>
			<fl:Object IDRef="ID_CRE_LogicalRef_143.1"/>
			<fl:Object IDRef="Port_Logical on subL"/>
		</fl:TargetPath>
		<fl:SourceCtx IDRef="ID_YFP_ROOT_Product"/>
		<fl:SourcePath>
			<fl:Object IDRef=""/>
		</fl:SourcePath>
		<fl:TargetCtx IDRef="ID_YFP_Root_Logical"/>
		<fl:TargetPath>
			<fl:Object IDRef="ID_CRE_LogicalRef_143.1"/>
		</fl:TargetPath>
	</fl:ImplementLink>
	<fl:ImplementLink Custo="CRE_Implement" Type="PortPort" ID="ID_Cnx_sub-2" Name="Cnx_sub-2" Modeler="RFLPLMImplementConnection">
		<fl:ObjectAttribute InternalName="E_export_id" Value="CRE_ImplCnx_167" Type="String" Name="E_Export_id" Mandatory="Y"/>
		<fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Cnx_sub-2" Type="String" Name="Name" Mandatory="N"/>
		<fl:SourceCtx IDRef="YFP_ROOT_Product"/>
		<fl:SourcePath>
```

```
                                <fl:Object IDRef=""/>
                        </fl:SourcePath>
                        <fl:TargetCtx IDRef="YFP_Root_Logical"/>
                        <fl:TargetPath>
                                <fl:Object IDRef="ID_YFP_Sub_Logical2.1"/>
                                <fl:Object IDRef="Port_150"/>
                        </fl:TargetPath>
                        <fl:SourceCtx IDRef="ID_YFP_ROOT_Product"/>
                        <fl:SourcePath>
                                <fl:Object IDRef=""/>
                        </fl:SourcePath>
                        <fl:TargetCtx IDRef="ID_YFP_Root_Logical"/>
                        <fl:TargetPath>
                                <fl:Object IDRef="ID_YFP_Sub_Logical2.1"/>
                        </fl:TargetPath>
                </fl:ImplementLink>
                <fl:Reference Custo="CRE_Logical" H="117" ID="ID_YFP_Root_Logical" Name="YFP_Root_Logical" ImagePath="/Images/ID_YFP_Root_Logical_---.emf" Modeler="RFLVPMLogical" W="252">
                        <fl:ObjectAttribute InternalName="E_power_02" Value="0W" Type="Real" Name="Cooling power" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_01" Value="" Type="String" Name="E_Fem_Rep" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_03" Value="" Type="String" Name="E_Pub" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_02" Value="" Type="String" Name="E_Domain" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogRef_142" Type="String" Name="E_Export_id" Mandatory="Y"/>
                        <fl:ObjectAttribute InternalName="V_version" Value="---" Type="String" Name="Version" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_YFP_Root_Logical" Type="String" Name="Name" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        <fl:Port Custo="CRE_Logical" ID="ID_Port_Logical on RootL" Name="Port_Logical on RootL" Modeler="RFLVPMLogical">
                                <fl:ObjectAttribute InternalName="E_frc_01" Value="-1N" Type="Real" Name="Calculated max Force" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="E_str_01" Value="" Type="String" Name="Calculated Force direction" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="E_str_03" Value="" Type="String" Name="E_Pub" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogPort_43" Type="String" Name="E_Export_id" Mandatory="Y"/>
                                <fl:ObjectAttribute InternalName="V_Direction" Value="In" Type="Enumere" Name="Direction" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Port_Logical on RootL" Type="String" Name="Name" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        </fl:Port>
                        <fl:Instance X="110" Y="148" Custo="CRE_Logical" H="48" ID="ID_CRE_LogicalRef_143.1" Name="CRE_LogicalRef_143.1" IDRef="ID_YFP_Sub_Logical1" Modeler="RFLVPMLogical" W="72">
                                <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogInst_1" Type="String" Name="E_Export_id" Mandatory="Y"/>
                                <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_CRE_LogicalRef_143.1" Type="String" Name="Name" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        </fl:Instance>
                        <fl:Instance X="234" Y="147" Custo="CRE_Logical" H="48" ID="ID_YFP_Sub_Logical2.1" Name="YFP_Sub_Logical2.1" IDRef="ID_YFP_Sub_Logical2" Modeler="RFLVPMLogical" W="72">
                                <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogInst_113" Type="String" Name="E_Export_id" Mandatory="Y"/>
                                <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_YFP_Sub_Logical2.1" Type="String" Name="Name" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        </fl:Instance>
                        <fl:Connection Custo="CRE_Logical" ID="ID_Cnx_sub1-sub2" Name="Cnx_sub1-sub2" Modeler="RFLVPMLogical">
```

```
                                <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogCnx_29" Type="String" Name="E_Export_id" Mandatory="Y"/>
                                <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Cnx_sub1-sub2" Type="String" Name="Name" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                                <fl:Path>
                                        <fl:Object IDRef="ID_YFP_Sub_Logical2.1"/>
                                        <fl:Object IDRef="ID_Port_150"/>
                                </fl:Path>
                                <fl:Path>
                                        <fl:Object IDRef="ID_CRE_LogicalRef_143.1"/>
                                        <fl:Object IDRef="ID_Port_Logical on subL"/>
                                </fl:Path>
                        </fl:Connection>
                </fl:Reference>
                <fl:Reference Custo="CRE_Logical" H="336" ID="ID_YFP_Sub_Logical1" Name="YFP_Sub_Logical1" ImagePath="/Images/ID_YFP_Sub_Logical1_---.emf" Modeler="RFLVPMLogical" W="504">
                        <fl:ObjectAttribute InternalName="E_power_02" Value="0W" Type="Real" Name="Cooling power" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_01" Value="" Type="String" Name="E_Fem_Rep" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_03" Value="" Type="String" Name="E_Pub" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_02" Value="" Type="String" Name="E_Domain" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogRef_143" Type="String" Name="E_Export_id" Mandatory="Y"/>
                        <fl:ObjectAttribute InternalName="V_version" Value="---" Type="String" Name="Version" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_YFP_Sub_Logical1" Type="String" Name="Name" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        <fl:Port Custo="CRE_Logical" ID="ID_Port_Logical on subL" Name="Port_Logical on subL" Modeler="RFLVPMLogical" IDExposedInstance="ID_SignalInstanceType_21">
                                <fl:ObjectAttribute InternalName="E_frc_01" Value="-1N" Type="Real" Name="Calculated max Force" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="E_str_01" Value="" Type="String" Name="Calculated Force direction" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="E_str_03" Value="" Type="String" Name="E_Pub" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogPort_42" Type="String" Name="E_Export_id" Mandatory="Y"/>
                                <fl:ObjectAttribute InternalName="V_Direction" Value="In" Type="Enumere" Name="Direction" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Port_Logical on subL" Type="String" Name="Name" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        </fl:Port>
                        <fl:Instance Custo="CRE_Type" ID="ID_SignalInstanceType_21" Name="SignalInstanceType_21" IDRef="ID_SignalInstance_21" Modeler="RFLVPMSystemType">
                                <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_TypeInst_21" Type="String" Name="E_Export_id" Mandatory="Y"/>
                                <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_SignalInstanceType_21" Type="String" Name="Name" Mandatory="N"/>
                                <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                        </fl:Instance>
                </fl:Reference>
                <fl:Reference Custo="CRE_Logical" H="336" ID="ID_YFP_Sub_Logical2" Name="YFP_Sub_Logical2" ImagePath="/Images/ID_YFP_Sub_Logical2_---.emf" Modeler="RFLVPMLogical" W="504">
                        <fl:ObjectAttribute InternalName="E_power_02" Value="0W" Type="Real" Name="Cooling power" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_01" Value="" Type="String" Name="E_Fem_Rep" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_03" Value="" Type="String" Name="E_Pub" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_02" Value="" Type="String" Name="E_Domain" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogRef_356" Type="String" Name="E_Export_id" Mandatory="Y"/>
                        <fl:ObjectAttribute InternalName="V_version" Value="---" Type="String" Name="Version" Mandatory="N"/>
```

```
                    <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_YFP_Sub_Logical2" Type="String" Name="Name" Mandatory="N"/>
                    <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                    <fl:Port Custo="CRE_Logical" ID="ID_Port_150" Name="Port_150" Modeler="RFLVPMLogical" IDExposedInstance="ID_SignalInstanceType_22">
                        <fl:ObjectAttribute InternalName="E_frc_01" Value="-1N" Type="Real" Name="Calculated max Force" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_01" Value="" Type="String" Name="Calculated Force direction" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_str_03" Value="" Type="String" Name="E_Pub" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_LogPort_150" Type="String" Name="E_Export_id" Mandatory="Y"/>
                        <fl:ObjectAttribute InternalName="V_Direction" Value="Out" Type="Enumere" Name="Direction" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_Port_150" Type="String" Name="Name" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                    </fl:Port>
                    <fl:Instance Custo="CRE_Type" ID="ID_SignalInstanceType_22" Name="SignalInstanceType_22" IDRef="ID_SignalInstance_21" Modeler="RFLVPMSystemType">
                        <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_TypeInst_22" Type="String" Name="E_Export_id" Mandatory="Y"/>
                        <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_SignalInstanceType_22" Type="String" Name="Name" Mandatory="N"/>
                        <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                    </fl:Instance>
                </fl:Reference>
                <fl:Reference Custo="CRE_Type" ID="ID_SignalInstance_21" Name="SignalInstance_21" Modeler="RFLVPMSystemType">
                    <fl:ObjectAttribute InternalName="E_export_id" Value="CRE_TypeRef_21" Type="String" Name="E_Export_id" Mandatory="Y"/>
                    <fl:ObjectAttribute InternalName="V_version" Value="---" Type="String" Name="Version" Mandatory="N"/>
                    <fl:ObjectAttribute InternalName="PLM_ExternalID" Value="ID_SignalInstance_21" Type="String" Name="Name" Mandatory="N"/>
                    <fl:ObjectAttribute InternalName="V_description" Value="" Type="String" Name="Description" Mandatory="N"/>
                </fl:Reference>
            </fl:RFLPImportExport>
```

Table 23: XML representation of the logical and behavioural architecture of the PPS exported from Enovia V6

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:j.0="http://www.vinci-consulting.com/siemens/" >
 <rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/connection#1">
  <j.0:flowconnectionrevision_hasPorts rdf:resource="http://www.vinci-consulting.com/dssystemview/port#5"/>
  <j.0:flowconnectionrevision_revision>A</j.0:flowconnectionrevision_revision>
  <j.0:flowconnectionrevision_name>Cnx_sub1-sub2</j.0:flowconnectionrevision_name>
  <j.0:flowconnectionrevision_masterRef>#ID_Cnx_sub1-sub2master</j.0:flowconnectionrevision_masterRef>
  <j.0:flowconnectionrevision_hasMasterRef rdf:nodeID="A0"/>
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/flowconnectionrevision"/>
  <j.0:flowconnectionrevision_id>ID_Cnx_sub1-sub2</j.0:flowconnectionrevision_id>
  <j.0:flowconnectionrevision_hasPorts rdf:resource="http://www.vinci-consulting.com/dssystemview/port#4"/>
  <j.0:flowconnectionrevision_subType>Network</j.0:flowconnectionrevision_subType>
  <j.0:flowconnectionrevision_accessRefs>#id11</j.0:flowconnectionrevision_accessRefs>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A0">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/flowconnection"/>
  <j.0:flowconnection_subType>Network</j.0:flowconnection_subType>
  <j.0:flowconnection_name>Cnx_sub1-sub2</j.0:flowconnection_name>
  <j.0:flowconnection_id>ID_Cnx_sub1-sub2master</j.0:flowconnection_id>
  <j.0:flowconnection_catalogueId>ID_Cnx_sub1-sub2mastercatId</j.0:flowconnection_catalogueId>
  <j.0:flowconnection_accessRefs>#id11</j.0:flowconnection_accessRefs>
 </rdf:Description>
 <rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/reference#3">
```

```
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/product"/>
  <j.0:product_subType>Functionality</j.0:product_subType>
  <j.0:product_productId>ID_YFP_Root_LogicalprodId</j.0:product_productId>
  <j.0:product_name>YFP_Root_Logical</j.0:product_name>
  <j.0:product_id>ID_YFP_Root_Logical</j.0:product_id>
  <j.0:product_accessRefs>#id11</j.0:product_accessRefs>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A1">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_parentRef>#ID_CRE_LogicalRef_143.1occProdId</j.0:occurrence_parentRef>
  <j.0:occurrence_instancedRef>#ID_Port_Logical on subL</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>ID_Port_Logical on subLoccPortId</j.0:occurrence_id>
 </rdf:Description>
 <rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/instance2#3">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/productrevision"/>
  <j.0:productrevision_subType>FunctionalityRevision</j.0:productrevision_subType>
  <j.0:productrevision_revision>A</j.0:productrevision_revision>
  <j.0:productrevision_name>YFP_Sub_Logical2.1</j.0:productrevision_name>
  <j.0:productrevision_masterRef>ID_YFP_Sub_Logical2</j.0:productrevision_masterRef>
  <j.0:productrevision_id>ID_YFP_Sub_Logical2.1</j.0:productrevision_id>
  <j.0:productrevision_hasMasterRef rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#3"/>
  <j.0:productrevision_hasChild rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#5"/>
  <j.0:productrevision_accessRefs>#id11</j.0:productrevision_accessRefs>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A2">
  <j.0:productview_hasOccurrence rdf:nodeID="A3"/>
  <j.0:productview_hasOccurrence rdf:nodeID="A4"/>
  <j.0:productview_hasOccurrence rdf:nodeID="A5"/>
  <j.0:productview_rootRefs>id6</j.0:productview_rootRefs>
  <j.0:productview_hasOccurrence rdf:nodeID="A6"/>
  <j.0:productview_hasOccurrence rdf:nodeID="A1"/>
  <j.0:productview_ruleRefs>#id2</j.0:productview_ruleRefs>
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/productview"/>
  <j.0:productview_hasOccurrence rdf:nodeID="A7"/>
  <j.0:productview_primaryOccurrenceRef>id6</j.0:productview_primaryOccurrenceRef>
  <j.0:productview_hasOccurrence rdf:nodeID="A8"/>
  <j.0:productview_id>id4</j.0:productview_id>
 </rdf:Description>
 <rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/instance2#2">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/productrevision"/>
  <j.0:productrevision_subType>FunctionalityRevision</j.0:productrevision_subType>
  <j.0:productrevision_revision>A</j.0:productrevision_revision>
  <j.0:productrevision_name>CRE_LogicalRef_143.1</j.0:productrevision_name>
  <j.0:productrevision_masterRef>ID_YFP_Sub_Logical1</j.0:productrevision_masterRef>
  <j.0:productrevision_id>ID_CRE_LogicalRef_143.1</j.0:productrevision_id>
  <j.0:productrevision_hasMasterRef rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#3"/>
  <j.0:productrevision_hasChild rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#4"/>
  <j.0:productrevision_accessRefs>#id11</j.0:productrevision_accessRefs>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A8">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_parentRef>#ID_YFP_Sub_Logical2.1occProdId</j.0:occurrence_parentRef>
  <j.0:occurrence_instancedRef>#ID_Port_150</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>ID_Port_150occPortId</j.0:occurrence_id>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A9">
  <j.0:plmxml_hasProductRevision rdf:resource="http://www.vinci-consulting.com/dssystemview/instance2#2"/>
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/plmxml"/>
  <j.0:plmxml_hasFlowConnectionRevision rdf:resource="http://www.vinci-consulting.com/dssystemview/connection#1"/>
  <j.0:plmxml_xmlns>http://www.plmxml.org/Schemas/PLMXMLSchema</j.0:plmxml_xmlns>
  <j.0:plmxml_hasProductView rdf:nodeID="A2"/>
  <j.0:plmxml_language>en-us</j.0:plmxml_language>
  <j.0:plmxml_hasProductRevision rdf:nodeID="A10"/>
  <j.0:plmxml_hasTerminal rdf:resource="http://www.vinci-consulting.com/dssystemview/port#5"/>
```

```
  <j.0:plmxml_hasTerminal rdf:resource="http://www.vinci-consulting.com/dssystemview/port#4"/>
  <j.0:plmxml_hasTerminal rdf:resource="http://www.vinci-consulting.com/dssystemview/port#3"/>
  <j.0:plmxml_hasFlowConnection rdf:nodeID="A0"/>
  <j.0:plmxml_author>Teamcenter P9000.1.0.20120215.01 - Engineer, Ed@TC91 SiemensDC(-2079025999)</j.0:plmxml_author>
  <j.0:plmxml_hasProductRevision rdf:resource="http://www.vinci-consulting.com/dssystemview/instance2#3"/>
  <j.0:plmxml_schemaVersion>6</j.0:plmxml_schemaVersion>
  <j.0:plmxml_hasProduct rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#5"/>
  <j.0:plmxml_hasProduct rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#3"/>
  <j.0:plmxml_date rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2012-04-06T18:53:26.516Z</j.0:plmxml_date>
  <j.0:plmxml_hasProduct rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#4"/>
  <j.0:plmxml_time rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2012-04-06T18:53:26.516Z</j.0:plmxml_time>
</rdf:Description>
<rdf:Description rdf:nodeID="A10">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/productrevision"/>
  <j.0:productrevision_subType>FunctionalityRevision</j.0:productrevision_subType>
  <j.0:productrevision_revision>A</j.0:productrevision_revision>
  <j.0:productrevision_name>YFP_Root_Logical</j.0:productrevision_name>
  <j.0:productrevision_masterRef>#ID_YFP_Root_Logical</j.0:productrevision_masterRef>
  <j.0:productrevision_id>rootPRid</j.0:productrevision_id>
  <j.0:productrevision_hasChild rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#3"/>
  <j.0:productrevision_accessRefs>#id11</j.0:productrevision_accessRefs>
</rdf:Description>
<rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/port#5">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/terminal"/>
  <j.0:terminal_subType>Network_Port</j.0:terminal_subType>
  <j.0:terminal_portOn rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#5"/>
  <j.0:terminal_name>Port_150</j.0:terminal_name>
  <j.0:terminal_id>ID_Port_150</j.0:terminal_id>
  <j.0:terminal_accessRefs>#id11</j.0:terminal_accessRefs>
</rdf:Description>
<rdf:Description rdf:nodeID="A11">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/reference"/>
  <j.0:reference_type>connection</j.0:reference_type>
  <j.0:reference_occurrenceRef>#ID_Port_Logical on subLoccPortId</j.0:reference_occurrenceRef>
  <j.0:reference_id>ID_Cnx_sub1-sub2ID_Port_Logical on subL</j.0:reference_id>
</rdf:Description>
<rdf:Description rdf:nodeID="A7">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_parentRef>#id6</j.0:occurrence_parentRef>
  <j.0:occurrence_instancedRef>#ID_Cnx_sub1-sub2</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>ID_Cnx_sub1-sub2occConnId</j.0:occurrence_id>
  <j.0:occurrence_hasReference rdf:nodeID="A12"/>
  <j.0:occurrence_hasReference rdf:nodeID="A11"/>
</rdf:Description>
<rdf:Description rdf:nodeID="A4">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_parentRef>#id6</j.0:occurrence_parentRef>
  <j.0:occurrence_instancedRef>#ID_Port_Logical on RootL</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>ID_Port_Logical on RootLoccPortId</j.0:occurrence_id>
</rdf:Description>
<rdf:Description rdf:nodeID="A3">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_occurrenceRefs>ID_Cnx_sub1-sub2occConnId</j.0:occurrence_occurrenceRefs>
  <j.0:occurrence_occurrenceRefs>ID_Port_Logical on RootLoccPortId</j.0:occurrence_occurrenceRefs>
  <j.0:occurrence_occurrenceRefs>ID_YFP_Sub_Logical2.1occProdId</j.0:occurrence_occurrenceRefs>
  <j.0:occurrence_occurrenceRefs>ID_CRE_LogicalRef_143.1occProdId</j.0:occurrence_occurrenceRefs>
  <j.0:occurrence_instancedRef>#rootPRid</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>id6</j.0:occurrence_id>
</rdf:Description>
<rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/port#3">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/terminal"/>
  <j.0:terminal_subType>Network_Port</j.0:terminal_subType>
  <j.0:terminal_portOn rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#3"/>
  <j.0:terminal_name>Port_Logical on RootL</j.0:terminal_name>
```

```
  <j.0:terminal_id>ID_Port_Logical on RootL</j.0:terminal_id>
  <j.0:terminal_accessRefs>#id11</j.0:terminal_accessRefs>
</rdf:Description>
<rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/reference#4">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/product"/>
  <j.0:product_subType>Functionality</j.0:product_subType>
  <j.0:product_productId>ID_YFP_Sub_Logical1prodId</j.0:product_productId>
  <j.0:product_name>YFP_Sub_Logical1</j.0:product_name>
  <j.0:product_id>ID_YFP_Sub_Logical1</j.0:product_id>
  <j.0:product_accessRefs>#id11</j.0:product_accessRefs>
</rdf:Description>
<rdf:Description rdf:nodeID="A12">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/reference"/>
  <j.0:reference_type>connection</j.0:reference_type>
  <j.0:reference_occurrenceRef>#ID_Port_150occPortId</j.0:reference_occurrenceRef>
  <j.0:reference_id>ID_Cnx_sub1-sub2ID_Port_150</j.0:reference_id>
</rdf:Description>
<rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/port#4">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/terminal"/>
  <j.0:terminal_subType>Network_Port</j.0:terminal_subType>
  <j.0:terminal_portOn rdf:resource="http://www.vinci-consulting.com/dssystemview/reference#4"/>
  <j.0:terminal_name>Port_Logical on subL</j.0:terminal_name>
  <j.0:terminal_id>ID_Port_Logical on subL</j.0:terminal_id>
  <j.0:terminal_accessRefs>#id11</j.0:terminal_accessRefs>
</rdf:Description>
<rdf:Description rdf:about="http://www.vinci-consulting.com/dssystemview/reference#5">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/product"/>
  <j.0:product_subType>Functionality</j.0:product_subType>
  <j.0:product_productId>ID_YFP_Sub_Logical2prodId</j.0:product_productId>
  <j.0:product_name>YFP_Sub_Logical2</j.0:product_name>
  <j.0:product_id>ID_YFP_Sub_Logical2</j.0:product_id>
  <j.0:product_accessRefs>#id11</j.0:product_accessRefs>
</rdf:Description>
<rdf:Description rdf:nodeID="A5">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_parentRef>#id6</j.0:occurrence_parentRef>
  <j.0:occurrence_occurrenceRefs>ID_Port_150occPortId</j.0:occurrence_occurrenceRefs>
  <j.0:occurrence_instancedRef>#ID_YFP_Sub_Logical2.1</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>ID_YFP_Sub_Logical2.1occProdId</j.0:occurrence_id>
</rdf:Description>
<rdf:Description rdf:nodeID="A6">
  <rdf:type rdf:resource="http://www.vinci-consulting.com/siemens/occurrence"/>
  <j.0:occurrence_parentRef>#id6</j.0:occurrence_parentRef>
  <j.0:occurrence_occurrenceRefs>ID_Port_Logical on subLoccPortId</j.0:occurrence_occurrenceRefs>
  <j.0:occurrence_instancedRef>#ID_CRE_LogicalRef_143.1</j.0:occurrence_instancedRef>
  <j.0:occurrence_id>ID_CRE_LogicalRef_143.1occProdId</j.0:occurrence_id>
</rdf:Description>
      </rdf:RDF>
```

Table 24: Representation of the logical and behavioural architecture of the PPS transformed into in RDF1

APPENDIX XVI

The BDA architecture framework

The BDA Architecture Framework (BDA-AF) is a structured set of process and information models and reference guidelines, which enables CRESCENDO partners to share a common understanding of the objectives and concepts required to define and manage the Behavioural Digital Aircraft architecture and system of enabling capabilities. To provide the structured set of process and information models and reference guidelines, a four layered structure has been defined, as illustrated on Figure 253:

- Level 1: The Business (Process) Layer and architecture
- Level 2: The Functional (Logical) Layer and architecture
- Level 3: The Application (Tool) Layer and architecture
- Level 4: The (Information) Technology Layer and architecture

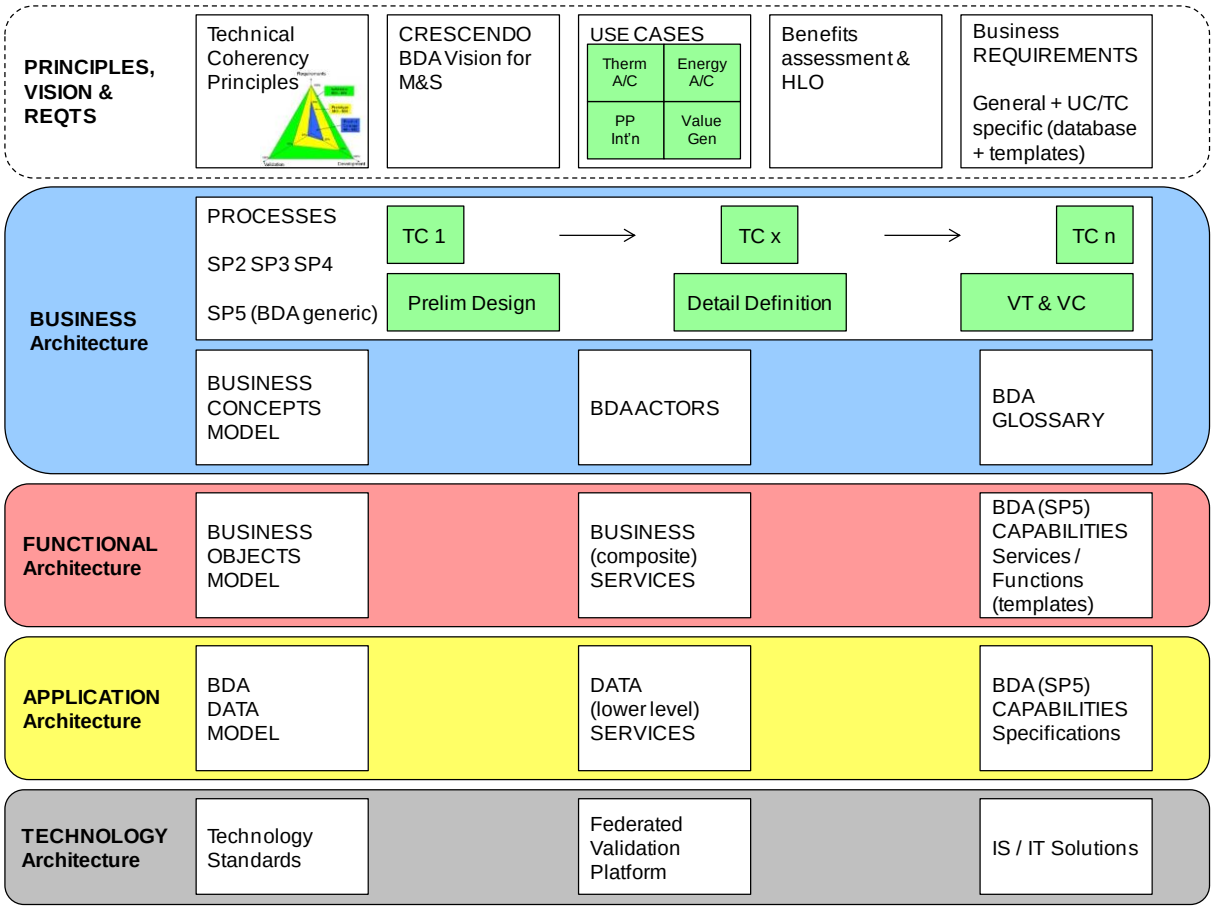


Figure 253: BDA Architecture Framework Layers and related artefacts

The layering enables a decoupling of the business processes, functions, applications and technologies supporting the BDA capability system. Therefore, when a layer is modified, the impacts on the same level and on the others will be effectively identified; which offers a better change control.

The BDA Information Model Hierarchy

A key aspect of the BDA is the management and sharing of information by the components of the BDA and the various actors that use the BDA. If the BDA is to be implemented as an open and modular

architecture, information models should be specified and based on standards. In particular, the following has been defined, with reference to the related layers of the BDA architecture framework:

- **The Business Concept model** i.e. the information requirements identified during the development of the processes that are to be supported by the BDA;
- **The Business Object model** i.e. the information that is generated by or consumed by BDA functions and services;
- **The BDA data model** i.e. the information that is managed by the BDA components.

The information models identified above are hierarchical with mappings between the layers in the hierarchy as shown in Figure 254 below.

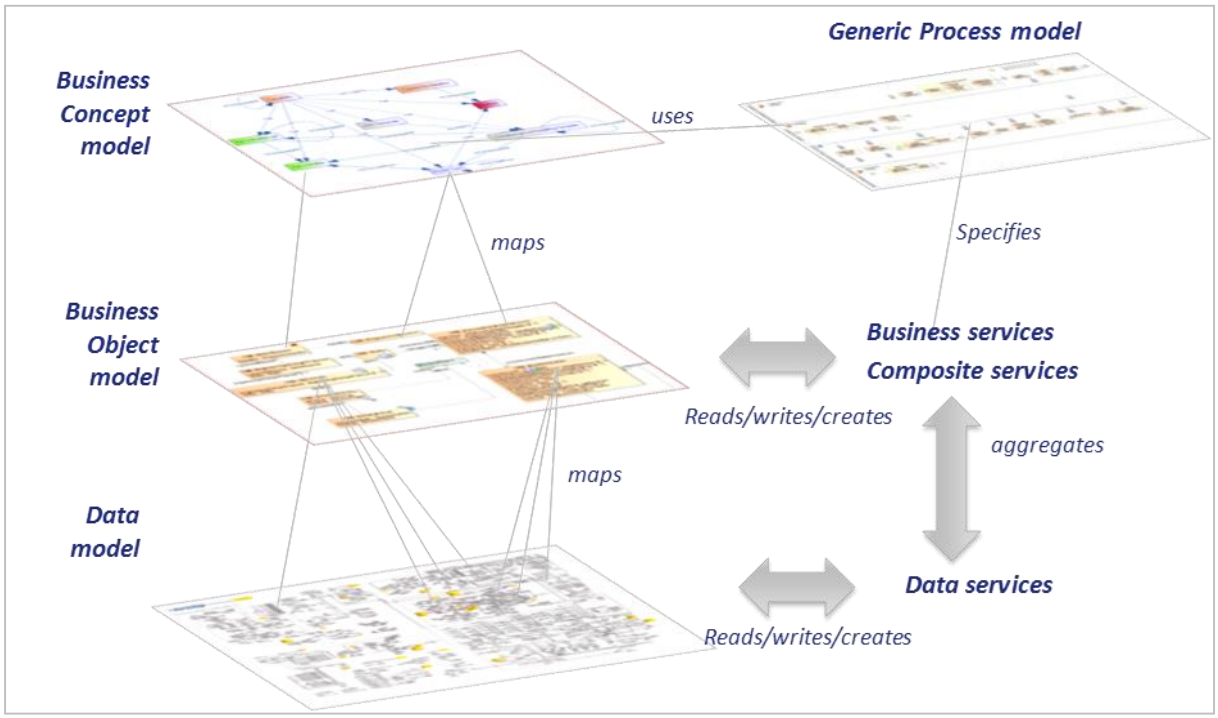


Figure 254: The information model hierarchy and how it relates to business processes and services.

BDA Business Concept model

The Business Concept Model (see Figure 255) is positioned in the Business Layer of the BDA-AF. The Business Concept Model is used to define information at a business level and is referenced by the process models that define how the BDA will be used. The definitions and terminology defining the information are expressed in the language used by business domain experts (the end users). The information is conceptual. In other words, the information does not provide sufficient detail for implementation, but rather acts as a high level specification and requirement for the lower level information layers.

The Business Concept Model does not need to have a formal mapping to the Business Object model. However the Business Object model has the responsibility to be able to represent the concepts and semantics expressed by the Business Concept model.

BDA Business Object model

The Business Object Model (see Figure 256) is positioned in the Functional Layer of the BDA-AF. The Business Object Model is derived from the Business Concept model (and influenced by the process models) and is also represented as an information model, but might have another structure.

From a Business Object model perspective, the Business Concept model serves as the requirements of the information and semantics that needs to be represented by a set of information entities and their behaviour which results in the higher level BDA Business services / functions.

Hence, the Business Object Model is a more detailed information model that documents the information that is consumed by or provided by the BDA functions and services.

The Business Object Model is also seen from the Data Model perspective as an abstraction, or aggregation of a set of entities governed by the Data Model. The Business Object Model is a strict mapping to a selected set of Data Model entities and defines how these Data Model entities are populated.

The Business Object model can be represented, for example, using UML class model with a definition, attributes and relationships.

BDA Data Model

The Data Model is positioned in the Application Layer of the BDA-AF. The Data Model is an integrated information model for representing the product data through life (from concept, through design, to in service and disposal). The Business Object Model maps to the Data Model.

The BDA Data Model specifies the data that is consumed or provided by Data Services. These services collect the data for the higher level Business or Composite Services.

In order to ensure an open and modular architecture, it is proposed that the BDA Data Model is mainly based on appropriate existing information standards, such as ISO 10303-239/233.

The Data Model can be represented, for example, using UML class model with a definition, attributes and relationships

BDA Business services (Functions)

The Business services are positioned in the Functional Layer of the BDA-AF.

The BDA Business services comprise methods and functionality that operates on larger chunks of Business Objects using business logic that cannot be expressed using the more fine-granular BDA Data services. For example, a Business Service (function) could be invoked and also invoke other external methods, or services. Some semi-automated orchestrations could be invoked as BDA Business services.

BDA Data Services

The Data Services are positioned in the Application Layer of the BDA-AF. The BDA Data services provide an implementation model based on the BDA Business Object model. The Data services provide CRUD (Create, Read, Update and Delete) functionality for the Business objects defined in the Business Object model.

In addition to the characteristics defined for each Business Object, the Data Services defines additional characteristics necessary for using the Business Objects in an implementation environment. For example, attributes for data provenance, internal system identification tokens, last updated, last modified, created by, created on etc.

The Data Services can be implemented using various technologies, for example by using Web Services, or any other XML Schema based technology since the Business Object model and the Business Services are defined independent from any communication protocol.

The overview diagrams for the Business Concept and Business Object Models are shown below.

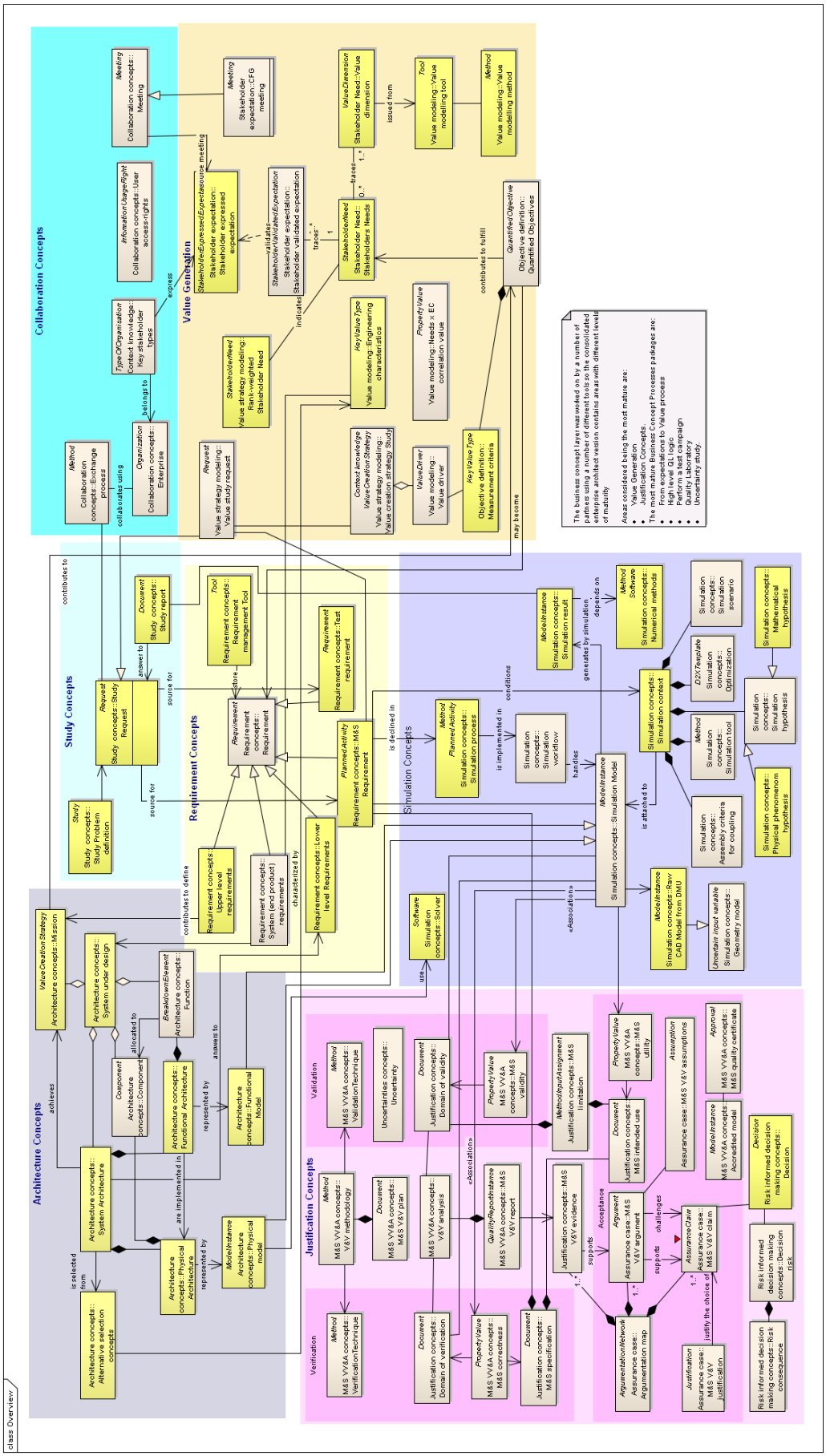


Figure 255: Business Concept Model Overview

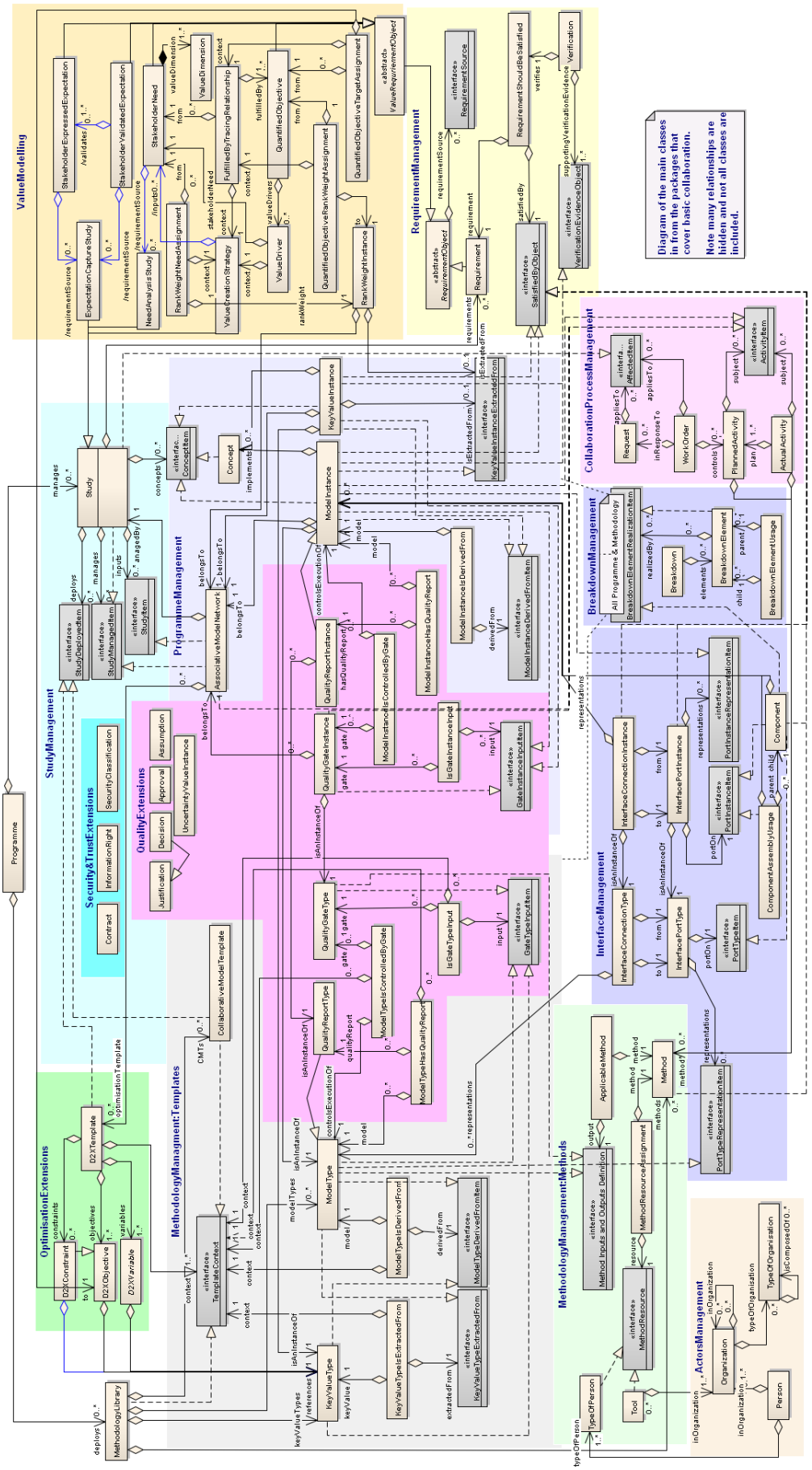


Figure 256: Business Object Model overview